VYSOKÁ ŠKOLA BÁŇSKÁ – TECHNICKÁ UNIVERZITA OSTRAVA
EKONOMICKÁ FAKULTA

KATEDRA SYSTÉMOVÉHO INŽENÝRSTVÍ

Návrh a implementace webové aplikace pro evidenci studentů

Design and Implementation of the Web Application for Students‘ Records

Student: Bc. Václav Homola
Vedoucí diplomové práce: Ing. Vítězslav Novák, Ph.D.

Ostrava 2015

VŠB - Technická univerzita Ostrava
Ekonomická fakulta
Katedra systémového inženýrství

Zadání diplomové práce

Student: **Bc. Václav Homola**

Studijní program: N6209 Systémové inženýrství a informatika

Studijní obor: 6209T025 Systémové inženýrství a informatika

Téma: Návrh a implementace webové aplikace pro evidenci studentů
Design and Implementation of the Web Application for Students' Records

Zásady pro vypracování:

1. Úvod
2. Metodická východiska tvorby webové aplikace
3. Analýza současného stavu evidence studentů
4. Návrh a realizace datového modelu a aplikační logiky
5. Závěr

Seznam použité literatury
Seznam zkratek
Prohlášení o využití výsledků diplomové práce
Seznam příloh
Přílohy

Seznam doporučené odborné literatury:

DEWSON, Robin. *Beginning SQL Server 2012 for Developer*. 3rd ed. New York: Apress, 2012. 697 p. ISBN 978-1-4302-3750-1.
HEFFELFINGER, R. David. *Java EE 6 Development with Netbeans 7*. Birmingham: Packt Publishing Ltd., 2011. 374 p. ISBN 978-1-849512-70-1.
ŠIMONOVÁ, Stanislava a Jan PANUŠ. *Databázové systémy I.* Pardubice: Univerzita Pardubice, 2007. 106 s. ISBN 978-80-251-1870-2.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Vítězslav Novák, Ph.D.**

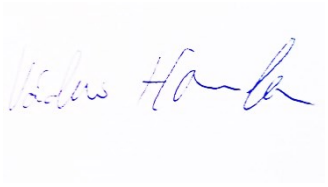
Datum zadání: 21.11.2014
Datum odevzdání: 25.04.2015

doc. Ing. Jana Hančlová, CSc.
vedoucí katedry

prof. Dr. Ing. Dana Dluhošová
děkanka fakulty

Místopřísežné prohlášení o samostatném vypracování diplomové práce
„Prohlašuji, že jsem celou práci, včetně všech příloh, vypracoval samostatně“.

Zároveň bych chtěl na tomto místě poděkovat svému vedoucímu diplomové práce, Ing. Vítězslavu Novákovi, Ph.D., za odborné vedení, cenné rady a připomínky, které mi pomohly k vypracování této diplomové práce.



……

Václav Homola

Datum odevzdání diplomové práce: 25. dubna 2015

Obsah

1 Úvod

Obsluha databází přes webového klienta v prostředí webové služby protokolu HTTP dnes patří k neodmyslitelným způsobům, jak přistupovat k datům, která jsou vzdálená klidně i stovky až tisíce kilometrů od koncových uživatelů. Existuje spousta způsobů a technologií, jak tento přístup k databázi zařídit a záleží čistě na projektovém manažerovi nebo architektovi, v čem hodlá svůj projekt realizovat.

Kancelář International Office na Ekonomické fakultě VŠB – TUO si každý semestr vede evidenci studentů, kteří se přihlašují do programu Erasmus. Je třeba evidovat jak jejich úspěšnost u zkoušek, což je nejdůležitější záznam, tak i informace o výjezdu. Jejich evidenční systém ovšem nesplňuje požadavky, které užíváním vyplynuly.

Cílem této diplomové práce tedy je navrhnout a vytvořit nové řešení evidenčního systému pro International Office, které využívá technologií, jež umožňují spravovat databázi pomocí webového prohlížeče, čili přistupovat k daným datům odkudkoli. Zaměřuje se na jednoduché a přehledné podání informací o studentech přihlášených do programu Erasmus.

Práce je rozdělena do tří kapitol, kdy první popisuje metodologická východiska práce, která jsou nezbytná pro návrh a implementaci vyvíjené části informačního systému a zabývá se důkladným popisem použitých metodik a technologií, pomocí nichž je aplikace vyvinuta. V této kapitole jsou tedy popsány technologie JavaServer Faces a JSF stránek, základní vlastnosti databáze, datové modelování, metodiky a procesy vývoje softwaru a v neposlední řadě modelování systému pomocí UML.

Následující, třetí, kapitola popisuje současný stav řešení, jeho nedostatky, chyby a návrh řešení nového, které je podrobněji popsáno v kapitole 4.

Jak už název napovídá, kapitola s názvem implementace aplikačního rozhraní v jazyku Java má za úkol popsat samotnou implementaci nového řešení, kdy se zprvu zabývá modelováním systému pomocí UML diagramů, návrhem nového datového modelu, transformací dat z původní databáze do nové a vytvoření prezentační vrstvy, sloužící ke komunikaci uživatele s databází.

2 Metodická východiska práce

V této kapitole jsou vymezeny základní pojmy, fakta a metody, které jsou nezbytné pro návrh a implementaci aplikace, čímž se rozumí databáze a uživatelské rozhraní, které nad danou databází pracuje. Důležité pro správný návrh aplikace je také UML a ER modelování.

2.1 Informační systém

Jelikož aplikace bude v podstatě modul do informačního systému, je třeba definovat, z čeho se skládá samotný informační systém.

Existuje celá řada definic informačního systému. Jedna z definic zní takto: „Informační systém lze definovat jako soubor lidí, metod a technických prostředků zajišťujících sběr, přenos, uchování, zpracování a prezentaci dat s cílem tvorby a poskytování informací dle potřeb příjemců informací činných v systémech řízení.“ (Tvrdíková, 2008).

Samotnou informaci lze popsat jako zprávu, která upřesňuje fakta o jevech nebo objektech reálného světa. V dnešní době je třeba potřebné informace efektivně získávat a pracovat s nimi. Pod pojmem systém si lze představit synonyma jako celistvost, organizace, struktura. Dnes je systém definován jako účelově definovaná množina prvků a vazeb mezi nimi a je chápán jako označení části reálného světa s charakteristickými vlastnostmi. Lze je rozdělit na přirozené, jež jsou takové systémy, které nejsou vytvořené člověkem a existují nezávisle na něm. Na druhou stranu umělé systémy člověk vytvořil a právě pod tyto systémy spadá i informační systém (Tvrdíková, 2008).

Jak již bylo napsáno výše, informační systém lze definovat jako soubor lidí, metod a technických prostředků zajišťujících sběr, přenos, uchování, zpracování a prezentaci dat s cílem tvorby a poskytování informací dle potřeb příjemců informací činných v systému řízení. Jiná definice zase tvrdí: „Informační systém je obecně podpůrný systém pro systém řízení. Jestliže chceme projektovat systém řízení jako takový, musíme znát, jaké jsou cíle, a informační systém řešit tak, aby tyto cíle podporoval.“ Tyto definice zahrnují člověka jako součást informačního systému a zmiňují se o míře potřeby příjemců informací. Informační systém se tedy skládá z těchto komponent:

- programové prostředky (software) – tvořené systémovými programy, které řídí chod počítače, práci s daty, komunikaci systému s reálným světem a aplikačními programy, které řeší úlohy zadávané uživateli,
- technické prostředky (hardware) – počítačové systémy různého druhu a velikosti, doplněné o potřebné periferní jednotky, které jsou v případě potřeby propojeny prostřednictvím počítačové sítě a napojeny na paměťový subsystém pro práci s velkými objemy dat,
- organizační prostředky (orgware) – tvořené souborem nařízení a pravidel, definujících provozování a využívání informačního systému a informačních technologií,
- lidská složka (peopleware) – řešení otázky adaptace a účinného fungování člověka v počítačovém prostředí, do kterého je vřazen,
- reálný svět (informační zdroje, legislativa, normy) – kontext informačního systému (Tvrdíková, 2008).

Tyto systémy lze taktéž rozdělit na transakční (TPS, Transaction Processing Systems), DSS (Decision Support Systems) systémy, jež jsou vyvinuty za účelem podpory rozhodování a systémy pro podporu vrcholového vedení (EIS, Executive Information Systems).

Transakční systémy tvoří základ informační pyramidy. Pod těmito systémy si lze představit aplikace pro řešení agentových úloh jako například fakturace, mzdy, účetnictví, inventarizace atd. Pracují na operativní úrovni a zajišťují základní procesy v organizaci.

Systémy pro podporu rozhodování (DSS) slouží k provádění analýz dat bez potřeby složitého ovládaní. Jsou určeny hlavně pro podporu středních složek managementu a jedná se o počítačovou podporu metod rozhodovací analýzy a operační systémové analýzy. Tyto systémy předpokládají, že uživatel rozumí podstatě metody a ví, jaká vstupní data do systému zadat.

EIS neboli manažerské aplikace zabezpečují vrchol řídící pyramidy. Slouží především vrcholovému vedení organizace, které se zajímá o informace z okolí organizace (technické inovace, trh, legislativa, konkurence). Mají přístup k externím datům, ale zároveň jsou napojeny na ostatní části systému firmy. Data v EIS mají vysokou vypovídající hodnotu (Tvrdíková, 2008).

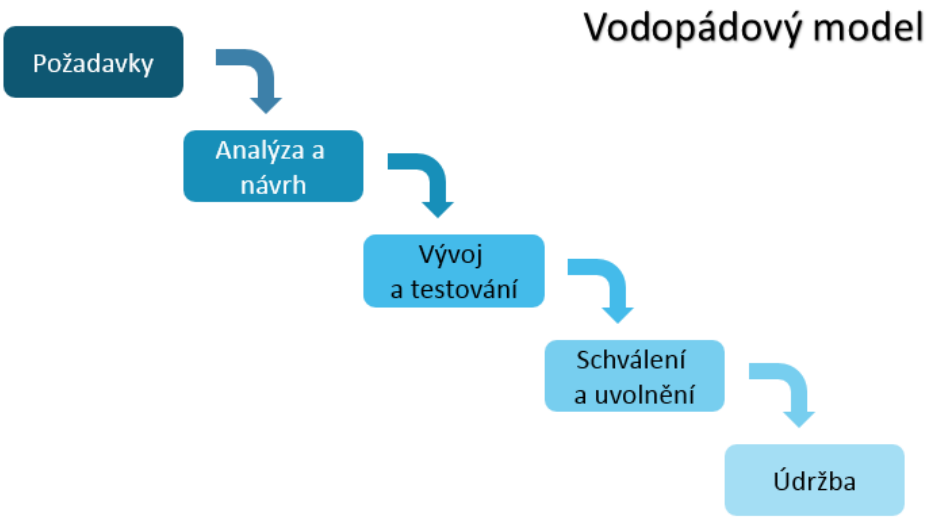
2.2 Metodika a procesy vývoje softwaru

V oblasti softwarového inženýrství jsou hlavní tři pojmy, které se velmi snadno zaměňují. Jedná se o pojmy metodologie, metodika a metoda, proto je vhodné tyto pojmy na začátku vyjasnit.

Metodologií se rozumí vědní disciplína, jež definuje a jinak analyzuje metodiky. Jedná se vesměs o „nauku o metodikách“. Metodikou je označován určitý komplexní postup a návod pro vývoj aplikace, kde se určují všechny fáze životního cyklu vývoje aplikace. Například metodika WebWAVE Development Process se zabývá fázemi vývoje internetové aplikace. Posledním pojmem zůstává metoda, která znamená konkrétní postup vedoucí k řešení problému.

Metodiky vývoje aplikací jsou jedněmi z nejdůležitějších produktů softwarového inženýrství. Za tu dobu, co systémové inženýrství existuje, bylo vyvinuto a navrhnuto spoustu metodik, jak co nejefektivněji vyvinout software podle požadavků. Mezi nejzákladnější modely, které je vhodné zmínit, je vodopádový model, spirálový model a agile metodika, která zvláště v současnosti zaznamenává velký rozmach.

Vodopádový model vznikl v roce 1970 a jedná se o sekvenční proces návrhu, který se používá v procesu vývoje softwaru. Tento model se skládá z jednotlivých fází jako je koncepce, zahájení, analýza, návrh, výroba, testování a produkce implementace, popřípadě udržování chodu. Model vznikl ve zpracovatelském průmyslu a stavebnictví a dalších průmyslových odvětvích, kde časté změny byly drahé a místy i nemožné. Protože v té době neexistovaly žádné metodiky na vývoj softwaru, vodopádový model se jednoduše adaptoval na tuto problematiku.

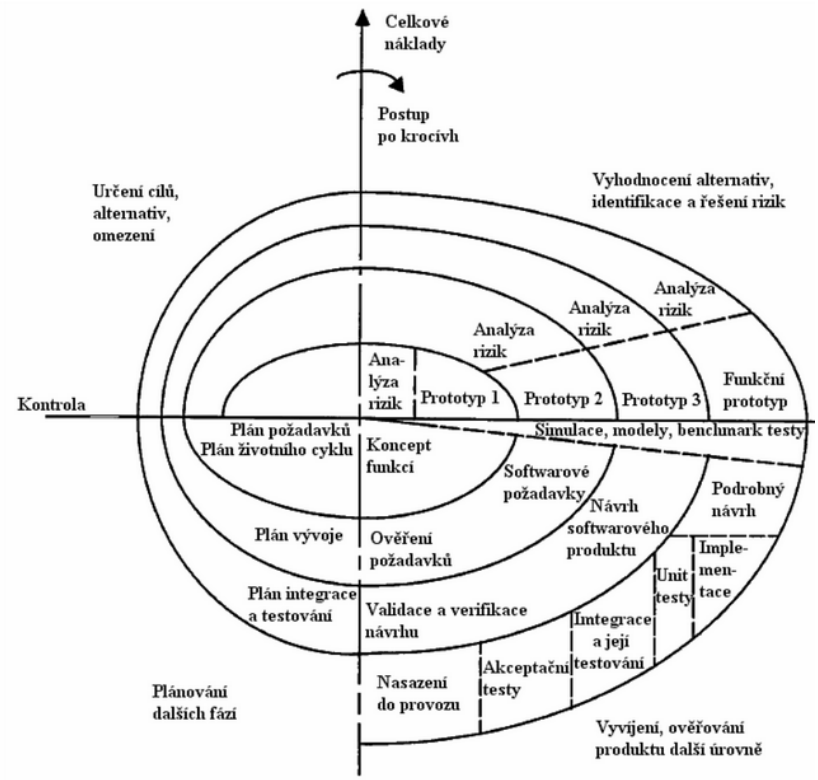


Obrázek 2.1 Vodopádový model (zdroj: managementmania.com)

Dalším modelem, který byl ve své době stěžejní, je spirálový model. Tento model poprvé definoval v roce 1986 Barry Boehm a pokrývá největší nedostatky vodopádového modelu. Spirálový model náleží do skupiny tzv. přístupů řízených riziky, čili postup do další fáze závisí na důsledné analýze všech rizik a možných problémů. Model je založen na principu iterace a zavádí opakovanou analýzu všech rizik. Model je vhodný pro větší projekty, protože se lépe vyrovnává s potenciálními modifikacemi. Hlavní myšlenkou tohoto modelu tedy je navazování nových částí na prověřený základ. Skládá se z několika kroků, které probíhají v iteracích, až dokud není produkt hotov. Tyto kroky jsou tedy:

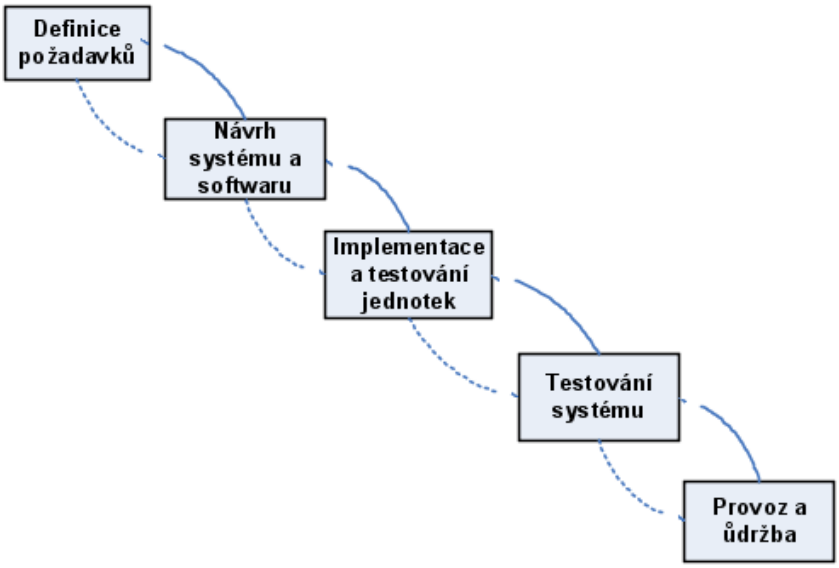
- určení cílů, alternativ, omezení,
- vyhodnocení alternativ, identifikace a řešení rizik,
- vývoj a verifikace další úrovně produktu,
- plánování následujících fází.

Po každé této fázi následuje testování, hodnocení a předání dílčích výsledků. Díky tomuto postupu je produkt testován ve svých raných fázích a mohou být zavčasu odhaleny chyby.



Obrázek 2.2 Spirálový model podle Boehma (zdroj: testovanisoftwaru.cz)

V dnešní době se tyto modely používají čím dál tím méně a nahrazuje je agilní přístup k vývoji systému, neboli agile. Tyto metodiky jsou založené na iterativním a inkrementálním vývoji. Díky tomuto přístupu je zajištěn rychlý vývoj softwaru a je možno reagovat na změny požadavků v průběhu vývoje. Správnost řešení se ověřuje jednoduše díky předložení jednotlivé iterace zákazníkovi, kdy je možno jednoduše řešení upravit díky zpětné vazbě, kterou zákazník poskytl. Agilní přístup našel uplatnění mimo programování také v Business Intelligence a v marketingovém plánování. Jedná se tedy o přesný opak vodopádového modelu. Na obrázku 2.3 lze vidět, jednotlivé iterace, kdy se od testování částečného řešení přechází zpátky k specifikaci a designu aplikace v další iteraci. Tento proces se opakuje, dokud produkt není hotový (Kadlec, 2004).



Obrázek 2.3 Grafické znázornění agilního přístupu (zdroj: statnice.dqd.cz)

2.3 UML

Jazyk UML (Unified Modeling Language, unifikovaný modelovací jazyk) je univerzální jazyk pro vizuální modelování systémů. Přestože je nejčastěji spojován s modelováním objektově orientovaných softwarových systémů, má mnohem širší využití, což vyplývá z jeho zabudovaných rozlišovacích mechanismů. Jazyk UML byl navržen proto, aby spojil nejlepší existující postupy modelovacích technik a softwarového inženýrství. Jako takový je explicitně navržen takovým způsobem, aby jej mohly implementovat všechny nástroje CASE (computer-aided software engineering). Zmíněná koncepce vychází z pochopení skutečnosti, že se rozsáhlé softwarové systémy obvykle bez podpory nástrojů CASE neobejdou. Diagramy vytvořené v jazyku UML jsou srozumitelné pro lidi, ale navíc je mohou snadno interpretovat i programy CASE (Arlow, 2007).

2.3.1 Způsoby využití UML jazyka

UML jazyk lze využít různými způsoby. Prvním je UML as Sketch, který pomáhá vývojářům v komunikaci mezi sebou. Lze využít u dopředného inženýrství, ale i u zpětného inženýrství. V dopředném inženýrství se z daných UML diagramů vytváří programový kód, kdežto u zpětného inženýrství je tomu naopak a z programového kódu jsou vytvořeny UML diagramy. Tohoto postupu se využívá, pokud je třeba popsat logiku a strukturu existujícího programu, ke kterému neexistuje dokumentace (Fowler, 2009).

Druhý způsob jak lze UML využívat je pro podrobný návrh (UML as blueprint). V dopředném inženýrství se předpokládá, že se pomocí něj vytvoří podrobný návrh, který programátoři následně kódují. Takovýto návrh se buď soustředí na určitou oblast, nebo může obsahovat všechny detaily. Tvorba těchto návrhu je velice náročná a komplikovaná, proto se pro usnadnění práce používají speciální CASE nástroje. UML as blueprint se liší od UML as sketch v podrobnosti návrhu, kdy blueprint je podrobnější (Fowler, 2009).

Výkonné CASE nástroje umožňují UML využít jako programovací jazyk, nicméně taková varianta se v praxi příliš nevyskytuje. Pomocí těchto nástrojů se vygeneruje základní kostra programu a detaily řeší už programátoři. Takové řešení lze za předpokladu detailně provedené specifikace softwaru jednoduše přenášet na různé platformy.

2.3.2 UML diagramy

UML 2 obsahuje a popisuje 13 typů oficiálních diagramů. Tyto diagramy lze rozdělit na dva typy, a to diagramy chování (Behavior Diagram) a diagramy struktur (Structure Diagram). Mezi hlavní se řadí diagram tříd, sekvenční diagram, diagram případů užití a diagram aktivit.

- Diagram tříd popisuje typy objektů v systému a různé druhy statických vztahů, které mezi nimi existují. Tyto diagramy ukazují vlastnosti a operace tří a omezení týkající se způsobu, jakým jsou objekty spojovány. Diagramy tříd tvoří páteř celého UML. Jelikož diagram tříd je v této práci zásadní, podrobnější popis je v následující podkapitole.
- Sekvenční diagram zachycuje chování jednoho scénáře. Ukazuje několik vzorových objektů a zpráv, které jsou předávány mezi těmito objekty v rámci daného případu užití. Sekvenční diagramy zobrazují spolupráci mezi objekty, nicméně mají mezery v definici chování (to zachycuje lépe stavový diagram).
- Diagram případů užití se používá k zachycení funkčních požadavků na systém. Případy užití popisují typické interakce mezi uživateli systému a systémem samotným a definují, jak je systém využíván. Jinými slovy, Use Case diagram znázorňuje chování popisovaného systému tak, jak jej vnímá uživatel. Toto můžeme popsat jako sled několika akcí, jež vedou k dosažení cíle. Případ užití nemá za úkol popsat, jak lze toho dosáhnout (Fowler, 2009).

- Diagram aktivit je využíván k popisování procedurální logiky, business procesů a work flow. Na rozdíl od vývojového diagrame je schopen zachytit paralelní chování.

2.3.3 Diagram tříd

Jak bylo řečeno výše, class Diagram, neboli diagram tříd zobrazuje statický pohled na systém, což znamená zejména třídy jako typy objektů, jejich obsah a statické vztahy, které mezi nimi existují. Diagram tříd může dále obsahovat balíčky a elementy chování, nicméně jejich dynamika je vyjadřována jinými diagramy. Při vytváření diagramu tříd je nutno vzít v potaz, zda je třeba vyjádřit požadavky na modelovaný software nebo získat popis designu. Díky těmto rozdílným požadavkům se dělí diagram tříd do tří úrovní:

- konceptuální – vytváří se za účelem analýzy požadavků, obsahuje business třídy, které slouží k modelování řešené oblasti a uvádí se zde pouze klíčové atributy a metody, jež mohou být také vynechány,
- designový – vychází z konceptuálního, který upřesňuje atributy, metody, datové typy atd. Navíc lze do tohoto modelu přidávat třídy uživatelského rozhraní a třídy obsluhující systémové události,
- implementační – znázorňuje grafické zobrazení implementovaného kódu.

Diagram tříd se skládá z několika prvků, jež jsou třídy, asociace, rozhraní a balíčky (Fowler, 2009).

2.4 Datové modelování

Tato podkapitola se zabývá definováním nezbytných pojmů pro vhodné datové modelování. I když určitý databázový model v současném řešení existuje a používá se, je třeba jej v práci v dalších kapitolách zrevidovat a vylepšit, aby vyhovoval novým požadavkům zadavatele. Aby byla databáze plně funkční, byla zajištěna integrita a odstraněna redundance dat, je třeba důkladně provést modelování databáze. Obecně se používá hlavně takzvaná tříúrovňová koncepce datového modelování, která zahrnuje sémantickou, konceptuální a logickou úroveň procesu. V této práci se autor zabývá pouze konceptuální a logickou úrovní.

2.4.1 Konceptuální datový model

Principem konceptuálního datového modelu, je jeho vyjádření pomocí grafických nástrojů. První metodou, která byla použita v tomto modelu, je metoda ER diagramu, kterou představil v roce 1976 Peter Chen. Tato metoda je vedle metody diagramu tříd, která tvoří součást metodiky UML (vytvořena v roce 1995 P. Boochem, J. Rumbaughem a I. Jacobsonem) nejpoužívanější v konceptuálním modelování.

Entita

Jedná se o objekt reálného světa, který je jednoznačně odlišný od ostatních objektů a je schopen nezávisle na ostatních existovat. Takovým objektem může být například Student, Univerzita, nebo Fakulta. Graficky lze entitu vyjádřit obdélníkem, v jehož horní části se nachází název entity a v dolní části jsou atributy. Entita může být silná, nebo slabá podle toho jak závisí na existenci jiné entity. Je-li entita silná, tak svým primárním klíčem nezávisí na žádné entitě, kdežto u slabé entity neexistuje žádný atribut, který by ji identifikoval (Šimonová a Panuš, 2007).

Vztah

Vztahem se rozumí vazba mezi dvěma, nebo více entitami. Nejběžnějším vztahem je tzv. asociativní vztah. Tento vztah reprezentuje asociace jedné nebo více entit. Každý tento vztah má tři charakteristiky.

První charakteristikou je stupeň a rozumí se jí počet entit, které se vyskytují v jednom vztahu. Stupeň může být unární (rekurzivní), kdy nejvyšší hierarchická úroveň je v takovéto vazbě tvořena jedním prvkem, který je svázaný s prvky o jednu úroveň níže. Proto každý prvek má vazbu na jeden prvek z vyšší úrovně. Toto neplatí pro prvek nejvyšší úrovně. Díky unární relaci lze vyjádřit vztahy typu nadřízený – podřízený. Dále jsou vztahy binární, kdy se ve vztahu vyskytují dvě entity a na stejném principu pracuje ternání až n-nární vztahy, které se ovšem vyskytují v praxi minimálně. Druhá charakteristika je kardinalita, která určuje počet výskytů entit v jednom výskytu vztahu.

- 1:1 – Každá entita je spojena vztahem nejvýše s jednou další entitou. Každý řádek primární tabulky lze svázat právě s jedním řádkem sekundární tabulky. V této kategorii vztahů se taky vyskytují takzvané částečné vztahy 1:0 a 0:1, které znamenají, že určitý řádek entity (výskyt entity) není přiřazen k žádnému výskytu entity v druhé tabulce,
- 1:N – Entita E je spojena vztahem s žádnou, nebo více entitami E_x. Tento vztah je velmi důležitý, jelikož eliminuje duplicitní data a nadbytečná data minimalizuje,
- M:N – Při tomto vztahu nejsou kladeny žádné podmínky a entity mezi sebou mohou být propojeny jakkoli, neexistuje žádné omezení. To obecně znamená, že více řádků entity může být spojeno s více řádky jiné entity. Žádný z relačních databázových systémů ovšem se vztahy M:N neumí pracovat, proto se tento typ vztahu v reálném světě uskutečňuje pomocí třetí spojovací tabulky, což v praxi znamená, že se vztah rozloží na dva vztahy 1:N.

Poslední charakteristikou asociativního vztahu je volitelnost. Ta určuje, zda je vztah povinný ze strany jedné entity nebo druhé, či nikoliv (Šimonová a Panuš, 2007).

2.4.1.1 *Atribut*

Jedná se o funkci nebo vlastnost, která přiřazuje entitám nebo vztahům hodnotu. Atributy jsou definovány jako složené nebo jednoduché, kde například v entitě Student mohou být atributy Id studenta, Jméno a Příjmení (Šimonová a Panuš, 2007).

2.4.1.2 *Klíč*

Klíč jednoznačně identifikuje entitu. Je to množina atributů, nebo jeden atribut jednoznačně určující výskyt entity, čímž se myslí jeden řádek tabulky. Klíč by měl být vhodně zvolen a splňovat určité kritéria, jimiž jsou:

- jednoznačnost – jedna hodnota identifikátoru by měla určovat právě jeden výskyt entity,
- úplnost – pro každý výskyt entity by měl existovat právě jeden identifikátor,
- minimální – klíč by neměl být tvořen žádnou podmnožinou, která je jednoznačná nebo úplná,
- stabilita – identifikátor nesmí během své existence měnit svou hodnotu.

Existují také různé druhy klíčů podle toho, jak se použijí a v jakém jsou vztahu:

- primární klíč – používá se v rámci entity a jednoznačně určuje její výskyt,
- kandidátní klíč – atributy, které lze zvolit jako primární klíče,
- cizí klíč – takhle se označuje atribut, který je v jiné entitě použit jako primární klíč. Tento klíč se primárně využívá k vyjadřování vztahů mezi entitami,
- alternativní klíč - pokud je jeden klíč zvolen jako primární, ale entita má další atributy, které jednoznačně určují entitní typ, ale nejsou zvoleny jako primární, nazývají se kandidátní,
- sekundární klíč – je to nejednoznačná množina atributů, která slouží k vyhledávání a třídění dat, které se nacházejí v entitě (Šimonová a Panuš, 2007).

2.4.2 Logický relační datový model

Nástup relační koncepce usnadnil technologický zlom vyvolaný nástupem osobních počítačů. Relační datové modelování přebírá některé konstrukty z konceptuální úrovně. Mezi ně patří atribut, doména a klíč, nicméně zavádí nový pojem relace. „Relace je dvourozměrná datová struktura, která je tvořena záhlavím a tělem relace. Záhlavím relace se rozumí množina dvojic (A_i, D_i), kde atribut A_i je přiřazen právě jedné doméně D_i, pro i = 1, 2, …, n a zároveň A_i musí navzájem být odlišná. Tělo relace je tvořeno množinou n -tic, které jsou množinami dvojic (A_i, v_{ji}), kde A_i je i -tý atribut a dále v_{ji} je j -tá hodnota z domény D_i pro j =1, 2, …, m, kde m je počet n -tic v množině; m je pak kardinalitou a n stupněm relace (pro n =1 se hovoří o unární relaci, pro n =2 o binární, atd.).“

Relace mají tyto základní vlastnosti:

- neexistence duplikátů n – tic,
- libovolné pořadí atributů,
- libovolné pořadí n – tic,
- nerozložitelnost hodnot atributů.

První tři vlastnosti znamenají, že tělo relace je v podstatě množinou atributů, poslední vlastnost ukazuje na atomičnost atributů. Tato vlastnost znamená, že atributy nelze dále rozložit, aniž by došlo ke ztrátě informace a výrazně odlišuje relační koncepci od předchozí síťové, ve které atribut může obsahovat skupiny údajů, a můžou se zde nacházet opakující se hodnoty. Pokud relace neobsahuje vícehodnotové atributy, lze tvrdit, že je v první normální formě (Kaluža a Kalužová, 2012).

Relační model také zahrnuje relační algebru a relační kalkul. Podle koncepce, kterou navrhl Codd, databáze získaly aparát predikátové logiky 1. řádu, které se vyrovnají díky své úspornosti ve vyjadřování relačnímu kalkulu. Relační algebra je v podstatě množina operací, které když se aplikují na relace, vracejí opět relaci. Pro tyto operace lze využít množinové operace jako součin, sjednocení, průnik a rozdíl (Šimonová a Panuš, 2007).

Aby byl výčet pojmů datového modelování úplný, je třeba si vysvětlit také pojem normalizace. Jedná se o proces, kdy dochází k odstraňování anomálií v datovém modelu. Datový model se postupně dekomponuje tak, že se atributy dělí do většího počtu relací, které jsou již normované. Relace se postupně transformují do vyšších normálních forem, až dokud není zajištěna další nedělitelnost.

Normální formy jsou v podstatě soubor pravidel, které by měla relace splňovat, aby práce s jejími daty byla jednodušší. Hlavní normální formy jsou čtyři:

- **první normální forma (1NF)** – Relace je v první normální formě tehdy, když všechny atributy jsou atomické. Tato forma vyplývá z definice relace a její základní vlastnosti o nedělitelnosti atributů,
- **druhá normální forma (2NF)** - Tabulka splňuje podmínku o druhé normální formě tehdy, splňuje-li první normální formu a každý neklíčový atribut je plně funkčně závislý na primárním klíči relace. Neklíčový atribut musí být závislý na celém primárním klíči,
- **třetí normální forma (3NF)** – Relace splňuje 3NF, jestliže je v 2NF a každý neklíčový atribut je netranzitivně závislý na primárním klíči (Šimonová a Panuš, 2007, str. 63).

- **Boyce - Coddova normální forma (BCNF)** – V této formě je relace, jestliže každý determinant funkční závislosti v relaci je zároveň kandidátním klíčem. Každá relace v BCNF je vždy v 3NF, což naopak neplatí. Jestliže nastane, že 3NF není v BCNF, zřejmě za to můžou tyto důvody:
    - v relaci jsou nejméně dva kandidátní klíče,
    - všechny kandidátní klíče jsou složené,
    - existuje překrývající se atribut v kandidátních klíčích.

Je vidět, že BCNF je striktnější než 3NF (Kaluža a Kalužová, 2012).

Je také definována čtvrtá a pátá normální forma, nicméně ty se vyskytují velmi zřídka, jelikož představují výjimečné situace, kdy primární klíč má tři a více složek. Obecně převládá názor, že v praxi stačí použít transformaci do BCNF.

2.5 Microsoft SQL Server

Začátky databázových systémů Microsoft se datují do roku 1992, kdy dosáhl ohromného úspěchu s produkty Access a Microsoft FoxPro. Aby Microsoft pronikl na trh DBMS, nechal si od společnosti Sybase vytvořit Sybase SQL Server, který se stal základem nadcházejícího Microsoft SQL Server, čímž dokázali konkurovat jak IBM, tak Oraclu. Spolu s firmami Sybase a Ashton Tate spolupracovali na vytvoření první verze SQL Serveru, která spatřila světlo světa v roce 1992 jako verze 4.2. První verze, kterou Microsoft vyvinul samostatně, byla verze 6.0.

Microsoft SQL Server je relační databázový systém (RDBMS), určený pro podnikové prostředí. Stejně jako jeho předchůdci, SQL Server 2012 obsahuje sadu rozšíření, aby se zvýšila produktivita Structured Query Langugage (SQL), což je standardní programovací jazyk vytvořený pro získávání informací z databáze. SQL Server 2012 na rozdíl od jeho předchůdce 2008 R2 nabízí nové možnosti, které jsou například:

- ColumnStore indexy – jedná se o read - only indexy, které shlukují data a zefektivňují zpracování velkých dotazů.
- PowerView – tento nástroj umožňuje generovat Business Intelligence (BI) reportů přímo v prostředí SQL Server (Mistry, 2012).

Novinek oproti 2008 R2 je mnoho, nicméně nemá smysl je v této práci vypisovat všechny. Autor pouze zmiňuje ty nejdůležitější a nejzajímavější.

Verze 2012 není nejnovější, v loňském roce vyšel dlouho očekávaný SQL Server 2014. Tato verze ale v této práci díky nízké rozšířenosti a počátečními problémy s kompatibilitou není použita.

2.5.1 T-SQL

T-SQL, neboli Transact-SQL, je proprietární rozšíření pro SQL od Microsoftu (dříve Sybase) a je používán právě v Microsoft SQL Server. T-SQL umožňuje řízení SQL dotazů, tvorbu podmínek pomocí IF, ELSE, CASE a další vymoženosti, jako pracování s proměnnými atd. Největší výhodou T-SQL je definování procedur a funkcí, které umožňují definovat dávky SQL, znovu využívat kód, definovat triggery a tím ulehčit aplikační logice od problémů, které se mohou řešit už na databázové úrovni. Administrátoři mohou díky systémovým procedurám administrovat server, nastavovat parametry SQL Serveru, vytvářet databáze, zálohovat nebo nastavovat práva. Těchto funkcí ovšem v této práci není využito.

2.6 JavaServer Faces

Aplikační logika vyvíjeného řešení, které je v této práci popisováno, je navrhnuto a kódováno pomocí technologie JavaServer Faces (JSF).

Tato technologie byla vyvinuta společností Sun Microsystems Inc. a je součástí Java 5 Enterprise Edition a vyšších verzí. Základním principem JSF je jednodušší vývoj webových aplikací, kdy vývojáři definují uživatelský interface pomocí XML anotací, kterým jsou předávána data k zobrazení nebo editaci z Java Beanů. Díky tomuto řešení je webová aplikace rozdělena na uživatelské rozhraní a aplikační logiku.

JavaServer Faces je tedy komponentově orientovaný webový Framework, kdy je stránka obsluhována jako celek, nicméně komponentový přístup JSF deleguje obsluhu stránky jako takové na jednotlivé komponenty, které se na webové stránce spravují samy. Stejně jako drtivá většina frameworků, je JSF postaveno na Servlet API a JSP (Tong, 2009).

Z vývojářského hlediska, JSF aplikace se skládá z několika xhtml stránek, které obsahují JSF anotace, JSF Managed Beany a konfigurační soubor faces-config.xml. V JSF 1.x byl faces-config.xml povinný, nicméně v JSF 2.0 a výš byly uvedeny určité konvence, které redukují důležitost tohoto souboru (Heffelfinger, 2011).

2.6.1 PrimeFaces

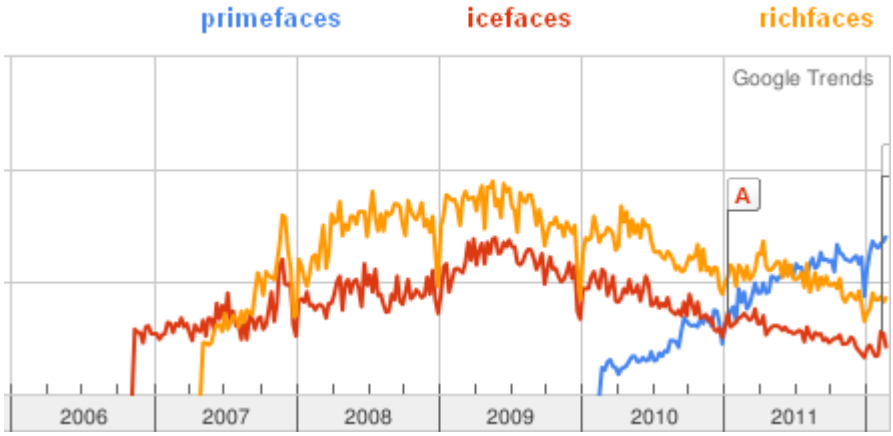
PrimeFaces je open source framework User Interface (UI) komponent pro JavaServer Faces, kterou vytvořil PrimeTek v Turecku. Vývoj PrimeFaces byl zahájen na konci roku 2008, kdy nahradily YUI4JSF knihovnu. Tato knihovna byla složena z JSF komponent postavených na YUI knihovny JavaScriptu. Na začátku roku 2009 byla YUI4JSF zrušena ve prospěch PrimeFaces. Od jeho vydání je tento Framework silně podporován společností Oracle, a to zejména v rámci NetBeans IDE, ve kterém je vyvíjena i aplikace řešená v této práci.

Framework PrimeFaces využívá mnoho známých společností k vývoji vlastních webových aplikací. Výčtem je možno zmínit například Tieto, BMW, Volkswagen, Cisco, Lufthansa a další mezinárodní společnosti (PrimeFaces, 2014).

Podrobnosti o využití JSF a frameworku PrimeFaces je popsáno níže v kapitole „Návrh a realizace datového modelu a aplikační logiky“, která se zabývá samotným technologickým aspektem aplikace, což zahrnuje důkladný popis metod, tříd a samotného aplikačního rozhraní.

Když se řekne Faces, vybaví se soubory XML anotací, kterými se specifikují odkazy na Java Beans v aplikačním serveru. Při psaní těchto XML souborů jsou využívány právě tyto anotace, importované z Tag Library Description souborů, které jsou součástí nějaké JSF knihovny. Sun Microsystems definoval vlastní knihovnu obsahující základní objekty, nicméně později se začaly rozšiřovat určité nástavby a rozšíření, mezi které patří právě PrimeFaces, které se liší svým zápisem, kdy místo <h:outputText>, definující textové pole, se píše <p:outputText>. Tyto objekty mají vlastní stylování, ale jejich atributy jsou víceméně totožné s klasickými JSF anotacemi. Některé anotace objevující se v PrimeFaces se nevyskytují v původních JSF knihovnách, například pro drag and drop, speciální grafy atd.

PrimeFaces, který je relativně nový framework, začíná být nejpoužívanějším pro tvorbu prezentační vrstvy, kdy v poslední době výrazně předstihuje právě zmiňované IceFaces a RichFaces. Ostatně tento trend lze krásně vidět na níže znázorněném grafu 2.4.



Obrázek 2.4 Trend používání frameworků (zdroj: masterthebоss.com)

V každé xhtml stránce, která je v aplikaci vytvořena pomocí frameworku PrimeFaces, je důležitá hlavička, jež udává, které anotace lze použít v samotném těle stránky. Lze vidět, že je použita klasická „h“ anotace, PrimeFaces „p“ a facelets „ui“ anotace.

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:p="http://primefaces.org/ui"
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets">
```

Příklad 2.1 Hlavička xhtml stránky

2.7 Java Persistence API

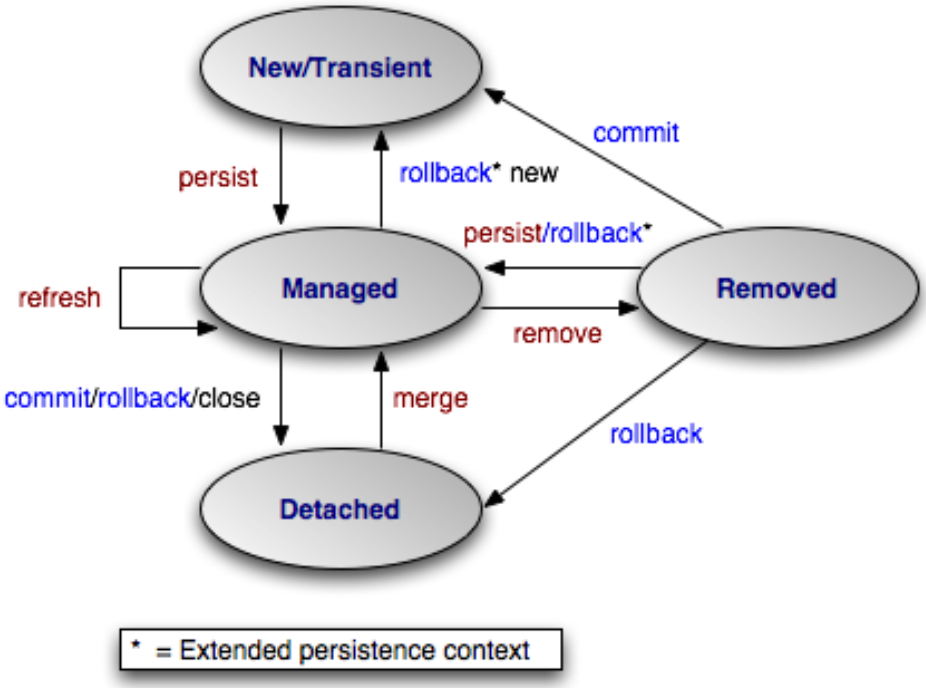
Java Persistence API (JPA) je objektově-relační mapovací technologie, která umožňuje aplikacím spravovat data mezi javovskými objekty a relační databází tak, že pro vývojáře je velmi transparentní a jednoduchá. To znamená, že je možno v projektech aplikovat JPA tak, že se vytvoří sada tříd (entit), které odrážejí datový model. Aplikace poté může přistupovat k těmto entitám tak, jako by se jednalo o přímý přístup k databázi. Použití JPA přináší několik výhod:

- vlastní dotazovací jazyk JPQL, jenž je podobný klasickému SQL, sloužící k vytváření statických, ale i dynamických dotazů. JPQL lze použít nad kteroukoli databází, kteréhokoli výrobce, ať už se jedná o MS SQL Server, Oracle, či MySQL a další,
- odpadává povinnost psaní kódu na nízké úrovni, které je náchylné k chybám v JDBC a SQL kódu,
- JPA poskytuje služby pro ukládání do mezipaměti dat a optimalizaci výkonu (NetBeans, 2014).

2.7.1 Entitní třídy

Každá entita má svůj stav, ve kterém se nachází, který je určován instancí třídy EntityManager. Entita mění svůj stav vždy po provedení určitého úkonu EntityManagerem. Počáteční stav je vždy New/Transient, následující obrázek 2.5 vystihuje všechny stavy entity a hrany mezi nimi.

- transient představuje instanci mimo správu Java Persistence Api, například čerstvě vytvořená instance neuložena metodou persist(),
- managed znamená objekt pod správcem JPA, který definuje, že objekt je získán z databáze po zavolání metody persist(). Změny na jeho instančních proměnných jsou monitorovány a následně, pokud je to nutné, uloženy do databáze,
- detached objekt už není pod kontrolou EntityManageru. Do tohoto stavu se objekt dostává nejčastěji po uzavření správce entit a zpět pod správu nového EntityManageru lze objekt dostat pomocí metody merge().
- Removed je objekt, který neexistuje v podobě instance, ale z databáze byl odstraněn pomocí metody remove(). I v tomto případě se jedná o jeden databázový řádek (Tong, 2009).



Obrázek 2.5 Životní cyklus entity (zdroj: openjpa.apache.org)

Entity můžou být persistovány, ale musí splňovat určité vlastnosti:

- musí být anotována pomocí javax.persistence.Entity
- musí mít public nebo protected konstruktor, který navíc nesmí mít žádné parametry, nicméně může mít další konstruktory,
- nesmí být deklarována jako final, stejně jako její metody nebo proměnné,
- pokud Session Bean pracuje s instancemi této třídy a je typu Remote, musí entitní třída implementovat Serializable rozhraní,
- můžou dědit jak z entitní, tak z ne-entitní třídy a ne-entitní třídy mohou dědit z entitních tříd,
- atributy je nutno deklarovat jako private, protected, nebo package-private a lze k nim přistupovat pouze přes entitní metody jako gettery a settery.

Takové entity jsou spravovány objektem třídy EntityManager, který pokud je třeba jej získat v Session Bean, je třeba, aby byl deklarován atribut typu EntityManager s anotací PersistenceContext. Tento EntityManager může pracovat s objekty pomocí metod:

- persist(java.lang.Object entity) – slouží k uložení objektu do databáze (operace INSERT),
- remove(java.lang.Object entity) – slouží k odstranění objektu z databáze (operace DELETE),
- merge(T entity) – podobná operace jako UPDATE, kdy entita je persistována, ale následně je změněna a po operaci merge se tyto změny projeví v databázi,
- find(java.lang.Class<T> entityClass, java.lang.Object primaryKey) – vrací objekt v tabulce, který koresponduje s class a má primární klíč id, což je v podstatě operace SELECT (Oracle, 2007).

2.8 Ajax

Jedná se o zkratku Asynchronous JavaScript and XML a označuje se tak technologie vývoje webových aplikací měnící obsah svých stránek tak, že není nutno znovu kompletně načítat stránku, ale toto načítání je prováděno pomoci asynchronního zpracování webových stránek pomocí JavaScript knihovny (Tong, 2009).

2.9 NetBeans IDE

NetBeans IDE je komplexní freewarové vývojové prostředí, které poskytuje prvotřídní a komplexní podporu pro nejnovější technologie Java. Toto vývojové prostředí poskytuje uživatelsky příjemný interface, kdy zdůrazňuje zdrojový kód jak syntakticky, tak sémanticky. Další z důležitých doplňků jsou šablony kódu, tipy pro kódování a také refactoring nástroje. Editor nepodporuje pouze programovací jazyk Java, nýbrž také C/C++, XML, HTML, PHP, JSP, Javascript a mnohé další. NetBeans lze využívat na různých platformách, které podporují Javu, čili není problém toto vývojové prostředí využívat jak na Windows, Linuxu, či Mac OS. Díky rozšiřitelnosti lze do NetBeans doinstalovávat, nebo vytvářet pluginy, které lze později využívat pro vývoj aplikací (NetBeans IDE, 2011).

3 Analýza současného stavu evidence studentů

Tato kapitola nastiňuje základní problémy spojené se stávajícím řešením evidence studentů, se kterými se musí běžný uživatel dennodenně potýkat.

Návrh a implementace databázové aplikace je uskutečněn pro International Office na ekonomické fakultě. Toto oddělení má na starosti zahraniční činnost ekonomické fakulty. Díky International Office pedagogové a studenti mohou požádat o institucionální zabezpečení zahraničních aktivit díky uzavřeným smlouvám, podporu vedení fakulty, činnosti zahraničního oddělení a spolupráci s katedrami, ale hlavně finanční prostředky z programu Erasmus a z MŠMT ČR.

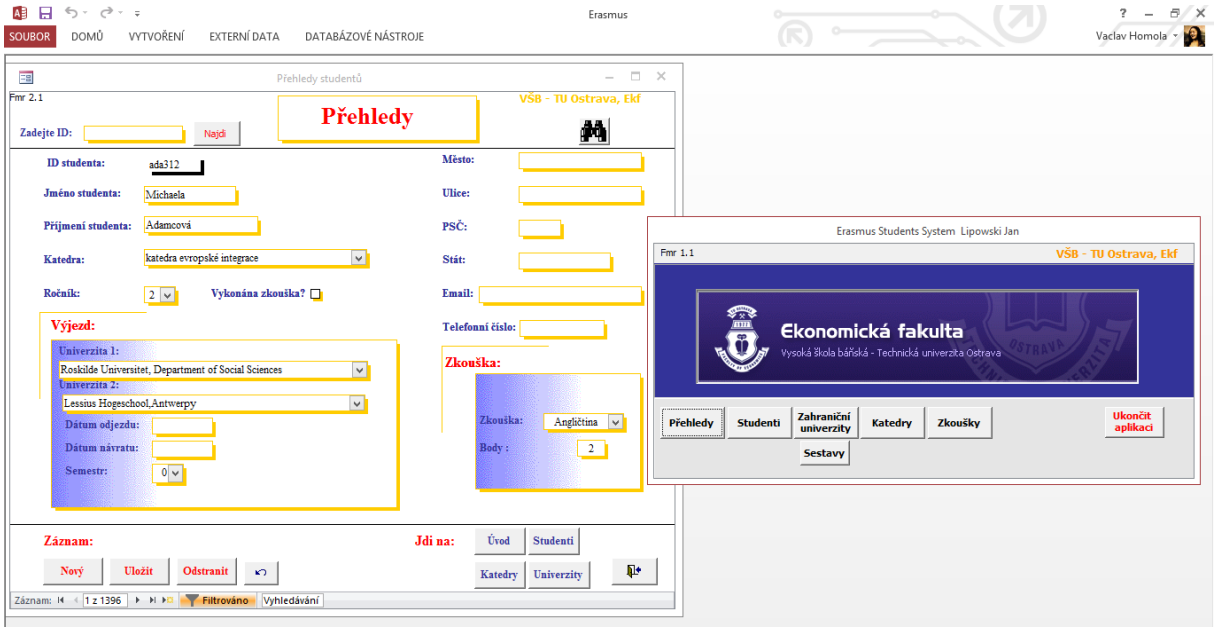
Oddělení zahraničních styků EkF VŠB-TU Ostrava dále zajišťuje a podporuje mezinárodní spolupráci EkF v souladu se zahraniční politikou VŠB-TUO, koordinuje spolupráci ekonomické fakulty se zahraničními univerzitami a institucemi, organizuje studentské a pedagogické výměnné pobyty, zejména přes program Erasmus, což jde ruku v ruce při organizování programů Evropské unie, mezi které právě Erasmus spadá.

3.1 Aktuální stav řešení

Současný stav evidence studentů je naprosto nevyhovující a zastaralý, který navíc nesplňuje určitá kritéria, jež by měla být při vývoji takové aplikace dodržována. Současné řešení je celé vytvořeno v prostředí Microsoft Access a to tak, že databáze, se kterou se pracuje, není nikde centrálně uložena na serveru, ale každá instance této aplikace má databázi vlastní, tudíž nastává při používání více uživateli chaos, kdy nikdo neví, která data jsou aktuální a která nikoliv.

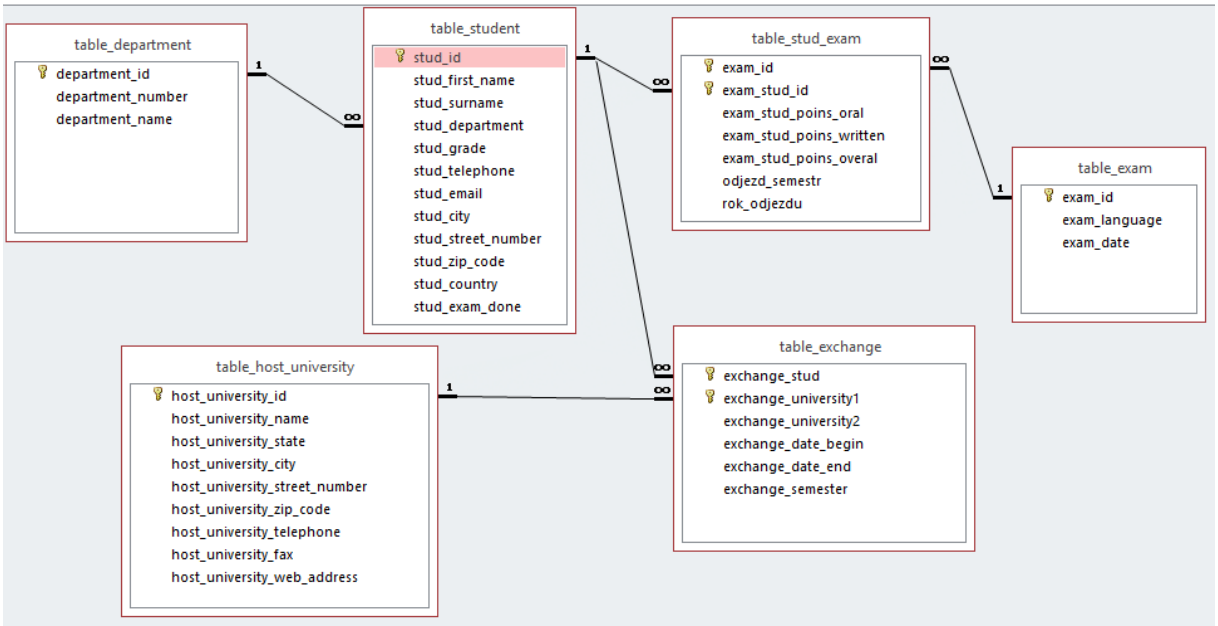
Dalším problémem je, že Access databáze přestává zvládat množství dat, které jsou zde uloženy, proto vznikl požadavek, přesunout databázi do nového prostředí, které by podobné množství dat bez problému zvládlo.

V neposlední řadě je zde také ne příliš příjemné uživatelské prostředí, které se tato práce také snaží vylepšit, aby se používání aplikace stalo jednodušším a příjemnějším, než je tomu doteď, viz obrázek 3.1.



Obrázek 3.1 Původní uživatelské rozhraní v Microsoft Access (zdroj: vlastní)

Na obrázku 3.2 je původní relační datový model. Už na první pohled jsou zde zřejmé základní nedostatky, které byly nastíněny v třetí kapitole, kde je analyzován současný stav.



Obrázek 3.2 Původní relační datový model (zdroj: vlastní)

Prvním problémem, který lze pozorovat, jsou nesmyslně zvolené primární klíče v tabulkách table_stud_exam a table_exchange, které slouží jako pomocné tabulky v M:N vztahu.

Druhá nesrovnalost, kterou je třeba vyřešit, jsou atributy v tabulkách, které evidentně mají být jinde. Příkladem takového atributu je stud_exam_done v tabulce table_student, který určuje, zda student vykonal zkoušku, nebo nikoliv. Ovšem zde není řešen případ, že student jde na více zkoušek. Díky tomuto lze pouze zjistit, zda student vykonal poslední zkoušku, na kterou je zapsán.

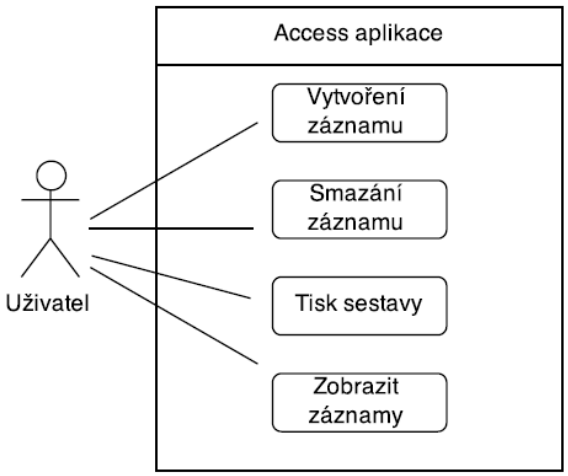
Tabulky table_stud_exam a table_exam nutí uživatele zapsat napřed termín zkoušky a ten poté přiřazovat studentům přihlášených na zkoušky, což je zbytečný krok na víc. Proto je vhodné tyto úkony minimalizovat, což řeší sloučení těchto tabulek v jednu.

Současné řešení umožňuje díky složenému primárnímu klíči pouze dva výjezdy, kdy tyto záznamy jsou zapsány ve dvou atributech, nicméně už se neřeší, který výjezd je zrovna aktuální a ke kterému se vážou záznamy o odjezdu a příjezdu. Proto k jednomu studentovi nemůže být zapsáno více výjezdů a tento problém je řešen pomocí mazání nejstaršího záznamu, díky čemuž nelze najít historická data.

3.2 Případ užití současného stavu

Vztah uživatele s aplikací v Accessu je znázorněn na obrázku 3.1, který potvrzuje, že existuje pouze jeden typ uživatele, který přistupuje ke všem činnostem v systému.

- **Uživatel** – tato osoba působí mimo systém, komunikuje s ním, přijímá a vkládá informace.
- **Typová činnost** – znázorňuje použití systému uživatelem.
- **Ohraničení** – vymezuje hranice popisovaného systému.
- **Relace** – vymezuje komunikaci mezi uživatelem a vnitřním případem užití.



Obrázek 3.3 Use Case diagram původní aplikace (zdroj: vlastní)

3.3 Návrh vylepšení

Nové řešení má tedy za úkol odstranit výše zmíněné problémy a nedostatky. V prvé řadě je tedy nejdůležitější umístit databázi na server, aby se odstranil první problém redundance databází. Databáze je vytvořena v prostředí Microsoft SQL Server 2012, kdy není problémem databázi migrovat a přesouvat dle potřeb. Navíc díky SQL Serveru odpadá problém s množstvím dat, jelikož tento software nemá problém zvládat desítky milionů záznamů v tabulkách. Dále je nutno upravit datový model tak, aby vyhovoval požadavkům zadavatele a nevznikaly v databázi duplicity a byly odstraněny problémy se špatně navrhnutými relacemi.

Samotné aplikační rozhraní je navrhnuto pomocí technologie JavaServer Faces, které je hojně využíváno k návrhu webových aplikací. Právě díky tomu, že aplikace bude implementována na webu, je možné k databázi přistupovat v podstatě odkudkoli, což byl také jeden z požadavků, které by nové řešení mělo splňovat. Celé uživatelské rozhraní je implementováno ve vývojovém prostředí Netbeans IDE 8.0.2.

4 Návrh a realizace datového modelu a aplikační logiky

Pro správný chod celého řešení je nutno navrhnout nový datový model, nad kterým bude operovat aplikační rozhraní. Dalším problémem je přesun stávajících dat z původní databáze do nové. Obojím se zabývají následující kapitoly. Následující část kapitoly se zabývá samotným vývojem aplikačního rozhraní pomocí JavaServer Faces.

4.1 Návrh nového datového modelu

Datový model je navržen po konzultaci se zadavatelem práce. Díky důslednému zpracování uživatelských požadavků není problém navrhnout datový model tak, aby byl okamžitě funkční a podchytil všechny problémy, které jsou v původním řešení. Tyto problémy jsou popsány v kapitole 3.1 a tato kapitola se zabývá jejími řešeními.

Tabulkám table_stud_exam a table_exchange, které sloužily původně jako pomocná tabulky k M:N vztahu, jsou zrušeny složené primární klíče a jsou zde zavedeny generované sekvence sloužící jako primární klíče, které jednoznačně identifikují jeden záznam v tabulce. Druhou možností je ponechat tyto tabulky bez klíče, nicméně tenhle způsob řešení se jeví k pozdějšímu vývoji aplikační a prezentační vrstvy jako nevyhovující.

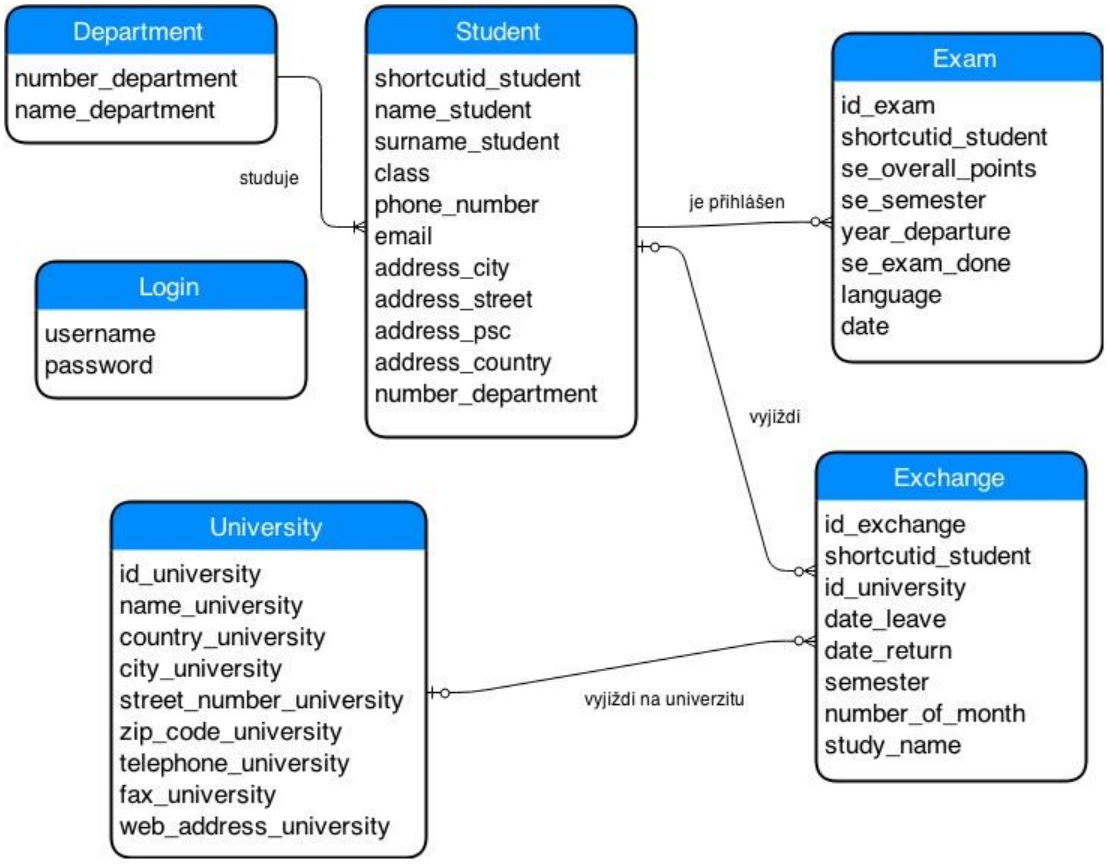
Aby bylo možno ke každé zkoušce a studentovi zjistit, zda ji vykonal či nikoliv, atribut stud_exam_done je nutno přiřadit tabulce se záznamy zkoušek, jelikož v tabulce table_student nemá žádný význam. Jak již bylo řečeno v kapitole 3.1, slouží pouze k zápisu o vykonání nejnovější zkoušky.

Tabulky table_stud_exam a table_exam jsou sloučeny v jednu, jelikož zapisovat zvlášť datum zkoušky a pak toto datum zvlášť přiřazovat k určitému studentovi je zbytečný krok navíc. Proto je vhodné tyto úkony minimalizovat, což řeší právě sloučení tabulek. Podrobnější popis tohoto úkonu je popsán níže.

Dalšími požadavky, které zadavatel požaduje, aby aplikace splňovala, jsou možnosti výjezdu na více univerzit. Toto řeší odstranění složeného primárního klíče a zavedení generované sekvence jako identifikátoru řádku, díky čemuž tabulka poslouží jako pomocná tabulka ve vztahu M:N mezi tabulkou univerzit a studentem a díky čemuž je možno zapisovat libovolné množství výjezdů ke studentům.

4.1.1 Konceptuální datový model

Konceptuální model zobrazuje entity a znázorňuje jejich vztahy mezi sebou. To znamená, že určuje kardinalitu mezi jednotlivými tabulkami, čili vztahy 1:N, 1:1 nebo M:N. Dále specifikuje volitelnost vztahů, generalizaci a specializaci. Tyto vztahy se obvykle znázorňují v E-R diagramu pomocí základních komponent jako entita a vztah s jeho vlastnostmi. Obrázek 4.1 specifikuje konceptuální model pomocí E-R diagramu a notace Crow's foot. Tento model nemá za úkol řešit cizí klíče, propojení tabulek ani jinou konkrétní databázovou koncepci. Tato struktura je nastíněna až posléze v logickém relačním datovém modelu.



Obrázek 4.1 E-R diagram datového model (Crow's foot) (zdroj: vlastní)

4.1.2 Relační datový model

Relační datový model na rozdíl od konceptuálního už řeší entity a vztahy tak, aby bylo možno jej transformovat do jazyka SQL. Rozdíl mezi entitou a relací je ten, že v relaci se už řeší primární klíče a cizí klíče určující vztahy mezi relacemi. Vztah 1:N je zde zapsán tak, že primární klíč jedné tabulky je přidán do tabulky druhé, kde je označen jako cizí klíč.

Vztah M:N se řeší pomocí tabulky s primárními klíči jedné i druhé tabulky, které jsou spolu v tomto vztahu a u vztahu 1:1 je třeba určit, zda je povinný nebo ne.

Pokud je povinný na jedné straně, zapíše se obdobně jako 1:N, pokud je oboustranně povinný, tak lze obě entity v tomto vztahu sloučit do jedné relace. Poslední dva jmenované vztahy se v řešeném modelu nevyskytují.

Struktura entit je znázorněna v následujících tabulkách pod odstavcem. Hodnoty „název“ a „datový“ jsou zřejmé, hodnota „klíč“ definuje, zda je daný atribut primárním (PK) či cizím klíčem (FK) a atribut „null“ nabývá vždy hodnot Ano, nebo Ne. Pokud je hodnota Ne, logicky atribut nesmí mít prázdný záznam a pokud Ano, nemusí být atribut v záznamu nutně vyplněn.

Relace student zůstává víceméně stejná jako v původním modelu, akorát atribut o vykonané zkoušce je přesunut do relace Exam, jelikož pouze tam dává smysl. Díky tomu lze zaznamenávat u jednoho studenta status o vykonání zkoušky i u více záznamů, než pouze u jednoho.

Student			
Název	**Datový typ**	**Klíč**	**Null**
shortcutid_student	varchar(10)	PK	Ne
name_student	varchar(50)		Ano
surname_student	varchar(50)		Ano
class	int		Ano
phone_number	varchar(13)		Ano
email	varchar(50)		Ano
address_city	varchar(50)		Ano
address_street	varchar(50)		Ano
address_psc	char(5)		Ano
address_country	varchar(20)		Ano
number_department	varchar(10)	FK	Ano

Tabulka 4-1 Tabulka student

Relace se záznamy kateder zůstává víceméně stejná, jelikož se jedná pouze o číselník s hodnotami.

Department			
Název	**Datový typ**	**Klíč**	**Null**
number_department	varchar(10)	PK	Ne
name_department	varchar(50)		Ano

Tabulka 4-2 Tabulka se záznamy kateder

Největší úpravy jsou evidentní u relace Student_Exam. Tato relace je vytvořena sloučením tabulky se zkouškami a záznamy studentů na zkouškách z původní databáze. Díky sloučení odpadává povinnost napřed zadat novou zkoušku do databáze a až poté zapisovat ke každému studentovi tento záznam. Tento problém ovšem řeší až aplikační logika, nicméně i proto je důležité navrhnout vhodně datový model, aby nedocházelo k neočekávaným a nechtěným problémům při zápisu dat. Také je odstraněn složený primární klíč, díky kterému nebylo možno zapsat jednoho studenta na více zkoušek a přidán atribut se_exam_done, který slouží k zápisu, zda byla daná zkouška vykonána, či nikoliv.

Student_Exam			
Název	**Datový typ**	**Klíč**	**Null**
id_exam	int	PK	Ne
shortcutid_student	varchar(10)	FK	Ne
se_overall_points	int		Ano
se_semester	varchar(10)		Ano
year_departure	varchar(10)		Ano
se_exam_done	varchar(10)		Ano
language	varchar(20)		Ano
date	date		Ano

Tabulka 4-3 Tabulka vykonaných zkoušek

Jako v předchozím případě byl z totožných důvodů odstraněn složený klíč a přidán atribut id_exchange, který slouží jako jednoznačný identifikátor záznamu. Ruku v ruce s odstraněním primárního složeného klíče jde i smazání původního atributu id_university2, který sloužil k přidávání záznamů k druhému výjezdu téhož studenta. Takovéto polovičaté řešení už není po úpravách třeba.

Exchange			
Název	**Datový typ**	**Klíč**	**Null**
shortcutid_student	varchar(10)	FK	Ne
id_university	int	FK	Ne
date_leave	date		Ano
date_return	date		Ano
semester	int		Ano
number_of_month	int		Ano
study_name	varchar(50)		Ano
id_exchange	int	PK	Ne

Tabulka 4-4 Tabulka s výjezdy studentů

Relace se záznamy univerzit nedosáhla žádných změn, jelikož stejně jako v případě relace s katedrami, jedná se pouze o číselník univerzit, do kterých studenti mohou vyjet v rámci programu Erasmus.

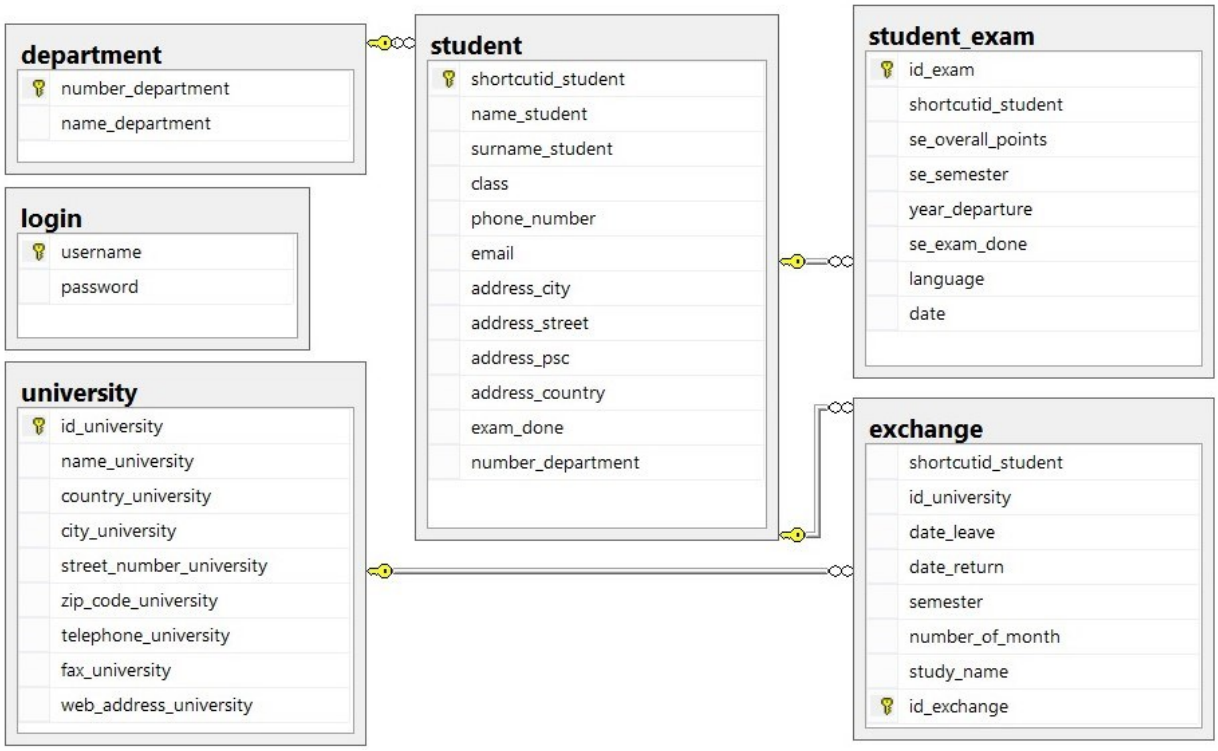
University			
Název	**Datový typ**	**Klíč**	**Null**
id_university	int	PK	Ne
name_university	varchar(100)		Ano
country_university	varchar(50)		Ano
city_university	varchar(50)		Ano
street_number_university	varchar(50)		Ano
zip_code_university	varchar(50)		Ano
telephone_university	varchar(50)		Ano
fax_university	varchar(50)		Ano
web_address_university	varchar(50)		Ano

Tabulka 4-5 Číselník univerzit

Takto určené relace by měly odstranit veškeré problémy, které se při užívání staré databáze vyskytují. Pro lepší přehled jsou relace ještě znázorněny v relačním diagramu, viz obrázek 4.2, kde jsou také zvýrazněny vztahy mezi jednotlivými relacemi.

Velice důležitá je tabulka login, ve které jsou uloženy přihlašovací údaje k aplikaci, přesněji v atributech username a password, které mají datový typ varchar(100) a jako primární klíč je použit atribut username. Tabulka není spojena žádným vztahem k dalším tabulkám.

Přihlašovací logika je popsána v pozdějších kapitolách. Relační diagram, ve kterém jsou znázorněny veškeré výše popsané tabulky, je vytvořen pomocí Microsoft SQL Server 2012.



Obrázek 4.2 Relační diagram (zdroj: vlastní)

4.2 Transformace dat

Do nově vytvořené databáze je nutno přesunout všechna data ze staré databáze tak, aby byla zachována integrita dat, neztratily se záznamy, v podstatě aby uživatel nepoznal, že došlo ke změně datové základny. Jelikož se datový model do jisté míry změnil, je třeba provést určité úpravy, aby byly splněny všechny požadavky na transformaci dat.

4.2.1 Migrace struktury databáze a dat do prostředí MS SQL Server 2012

Protože je původní řešení vytvořeno v prostředí Microsoft Access, je třeba databázi přesunout do Microsoft SQL Server 2012. Ještě ve verzi Accessu 2010 bylo možno tento úkon provést přímo v tomto softwaru. Jelikož autor pracuje s verzí 2013, která už toto nepodporuje, je potřeba migraci databáze z Accessu do SQL Serveru provést pomocí externího nástroje. Pro tento účel byl vybrán nástroj vyvinut Microsoftem přímo pro tyto operace, který nese název SQL Server Migration Assistant for Access. Tento nástroj supluje export databáze z Accessu 2010.

Export je velice intuitivní a pomocí průvodce exportem dojde až ke konečné destinaci, kam má být databáze vyexportována. Po potvrzení se Access databáze zkopíruje přímo na server.

Po úspěšném přesunu databáze do SQL Serveru je logické, že databáze je prázdná. Aby bylo možno pracovat s daty a později na nich testovat správnost aplikační logiky, je vhodné tato data taktéž přesunout ze staré do nové databáze.

Migrace dat ze staré databáze do nové je provedena pomocí SQL příkazů a to konkrétně SELECT a INSERT. První krok je vybrat data z původní databáze pomocí klauzule SELECT tak, aby odpovídala přesně novému datovému modelu a tabulce v něm, do které budou data vložena a posléze se daný výběr dat vloží do tabulky pomocí klauzule INSERT INTO.

Některé tabulky jsou vloženy jednoduše 1 ku 1, čili se jedná v podstatě o bezproblémovou konverzi dat. Nicméně některá data se přes chyby v původním modelu musí určitým způsobem transformovat a i přesto některé chyby nelze odstranit. Další odstavce nastiňují danou problematiku.

4.2.1.1 *Tabulka University*

Tato tabulka slouží jako příklad insertu klasického případu 1 ku 1, kdy nebylo třeba žádné úpravy. Jediný problém nastává pouze v případě vkládání už existujících hodnot do IDENTITY sloupce, který slouží jako primární klíč. Klauzule IDENTITY slouží k automatickému generování hodnot sloupce s novým řádkem o jednu jednotku větší. Tento problém se odstraní pomocí klauzule SET IDENTITY INSERT, viz ukázku SQL kódu níže.

```
SET IDENTITY_INSERT [ERASMUS].[dbo].[university] ON
insert into [ERASMUS].[dbo].[university]
([id_university],[name_university],[country_university],[city_university],
[street_number_university],
[zip_code_university],[telephone_university],[fax_university],
[web_address_university])
select [host_university_id],[host_university_name],[host_university_state],
[host_university_city],[host_university_street_number],
[host_university_zip_code],[host_university_telephone],[host_university_fax],
[host_university_web_address]
from [AccessDatabase].[dbo].[table_host_university]
```

Příklad 4.1 Insert do tabulky university do tabulky university

4.2.1.2 *Tabulka Student_Exam*

V této tabulce je nutno řešit problém o vykonané zkoušce, kterýž záznam byl původně v tabulce student. Díky tomuto řešení bohužel není možno zpětně dohledat, zda student vykonal více zkoušek, ale pouze tu nejaktuálnější, jelikož při každé jedné zkoušce se status o vykonání přepsal.

```
insert into [ERASMUS].[dbo].[student_exam]
([shortcutid_student],[se_overall_points],[se_semester],[year_departure],
[language],[date])
select
B.[stud_id],[exam_stud_poins_overal],[odjezd_semestr],[rok_odjezdu],exam_language,exam
_date
from [AccessDatabase].[dbo].[table_stud_exam] A
inner join [AccessDatabase].[dbo].[table_student] B on
A.[exam_stud_id] = B.[stud_id]
inner join [AccessDatabase].[dbo].[table_exam] C on
A.exam_id=C.exam_id
```

Příklad 4.2 Insert do tabulky student_exam

4.2.1.3 *Tabulka Exchange*

V původním řešení byly atributy exchange_university1 a exchange_university2, které byly vytvořeny proto, že složený klíč nedovoloval zápis více než jednoho výjezdu téhož studenta. Proto vzniklo toto řešení, nicméně nový model tento problém odstraňuje díky insertu, který je pod tímto odstavcem. Napřed jsou vloženy záznamy o výjezdu zapsaných v exchange_university1 a až posléze ty záznamy, jejichž univerzita je v atributu exchange_university2. Díky tomuto problému nelze dohledat datum příjezdu a odjezdu prvního výjezdu, jelikož ten byl pokaždé přepsán novým, druhým výjezdem. Naštěstí zadavatel na těchto hodnotách příliš nebazíruje, takže se není třeba tímto problémem dále zabývat.

```
SET IDENTITY_INSERT [ERASMUS].[dbo].[exchange] ON
insert into
[ERASMUS].[dbo].[exchange]([shortcutid_student],[id_university],[date_leave],
[date_return],[semester]
)
select A.[exchange_stud],[exchange_university1],[exchange_date_begin],
[exchange_date_end],[exchange_semester]
from [AccessDatabase].[dbo].[table_exchange] A inner join [dbo].[student] B on
B.[shortcutid_student] = A.[exchange_stud]
insert into
[ERASMUS].[dbo].[exchange]([shortcutid_student],[id_university],[date_leave],
[date_return],[semester]
)
select A.[exchange_stud],[exchange_university2],[exchange_date_begin],
[exchange_date_end],[exchange_semester]
from [AccessDatabase].[dbo].[table_exchange] A inner join [dbo].[student] B on
B.[shortcutid_student] = A.[exchange_stud]
where [exchange_university2] <> 0
```

Příklad 4.3 Insert do tabulky exchange

4.2.2 Vytvoření pohledů v nové databázi

V samotné nové databázi jsou vytvořeny pohledy, které se v aplikačním rozhraní používají jako sestavy, které slouží k tisku. Tento způsob je nejjednodušší a plní požadavky, tudíž se autor uchýlil k tomuto řešení. Jelikož pohledy obvykle nemají, nebo nemusí mít primární klíč a řešení v aplikační části určité id vyžaduje u každé entity, je třeba vytvořit umělý sloupec nazvaný „id“, kde se pomocí funkce row_number() načítá hodnota, která se s každým řádkem zvětšuje. Jedná se v podstatě o obdobu identity column.

První pohled slouží k náhledu výjezdů studentů, kde je důležité id studenta, jeho jméno, příjmení, název univerzity, kam vyjíždí a rok odjezdu.

```
create view [dbo].[exchange_view] as
select cast(row_number() over (order by (select NULL)) as varchar(20)) as
id,e.[shortcutid_student],[name_student],[surname_student],[name_university],
year(date_leave) as year_departure
from [dbo].[student] s inner join [dbo].[exchange] e on
s.shortcutid_student=e.shortcutid_student
inner join [dbo].[university] u on
e.id_university=u.id_university
```

Příklad 4.4 Vytvoření pohledu výjezdů

Nejdůležitější součástí je pohled s vykonanými zkouškami, ať už úspěšně, či neúspěšně. Mimo údajů o studentovi je zde záznam, zda vykonal zkoušku, z jakého jazyka a kdy se zkouška konala. Důležité je zmínit, že datum je převedeno do datového typu varchar kvůli pozdějšímu filtrování v aplikační rovině.

```
create view [dbo].[student_exam_view] as
select cast(row_number() over (order by (select NULL)) as varchar(20)) as id,
s.[shortcutid_student],[name_student],[surname_student],[se_exam_done],[language],cast
([date] as varchar(30)) as Date,[se_overall_points]
from dbo.student s inner join dbo.student_exam se on s.shortcutid_student =
se.shortcutid_student
```

Příklad 4.5 Vytvoření pohledu výjezdů

Posledním pohledem je celkový přehled v podstatě všech záznamů v databázi, čili údaje o studentovi, jeho výjezdu a vykonané zkoušce. Jako primární klíč zde neslouží generované id jako v předchozích případech, ale id studenta, které je pokaždé jedinečné.

```
create view dbo.overview as
select s.shortcutid_student, s.name_student, s.surname_student, name_department,
class,
name_university,date_leave, date_return, semester, se.language, se_overall_points
from dbo.department d inner join dbo.student s on
d.number_department=s.number_department
inner join dbo.student_exam se on s.shortcutid_student=se.shortcutid_student
inner join dbo.exchange e on s.shortcutid_student=e.shortcutid_student
```

```
inner join dbo.university u on e.id_university=u.id_university
```

Příklad 4.6 Celkový přehled

4.2.3 DateDiff procedura

V jednom z požadavků zaznělo, že je třeba zjistit, kolik měsíců už studenti byli v zahraničí, jelikož existuje určitý limit. Samozřejmě tyto údaje v původní databázi chybí, proto je potřeba hodnoty dopočítat. To se děje pomocí kódu, znázorněném níže, který počítá rozdíl mezi datem odjezdu a příjezdu, aby uživatelé měli přehled o počtu měsíců na výjezdech přes Erasmus. Důležitá funkce je datediff, která počítá rozdíl mezi dvěma daty. Výsledkem je integer, jenž udává právě počet měsíců, proto celkové dělení třiceti. Problém nastává v případech, kdy uživatelé nevyplnili oba datumy (odjezd a návrat), v tomto případě bohužel není možné počet měsíců dopočítat.

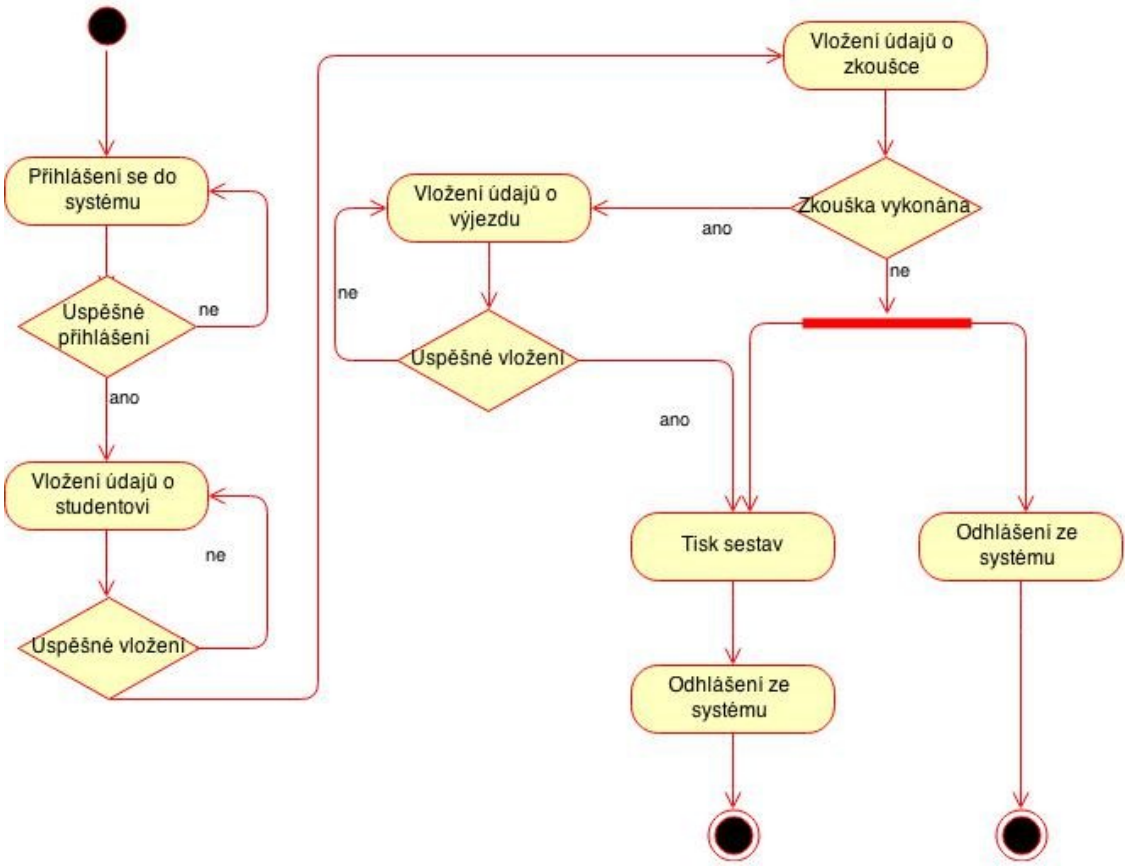
```
create procedure dbo.monthex
update [dbo].[exchange]
set [number_of_month] =
round((select case when datediff(d, date_leave, date_return)>30 then
datediff(d, date_leave,date_return)/30.0
else 0 end),0)
from [dbo].[exchange]
```

Příklad 4.7 Výpočet rozdílu mezi dnem odjezdu a návratu

4.3 Implementace aplikačního rozhraní v jazyku Java

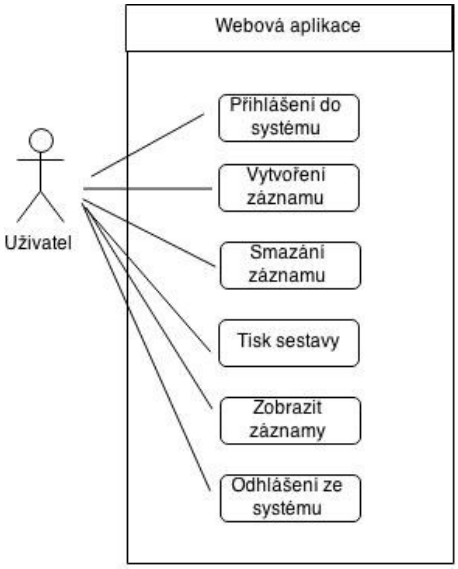
V předchozí kapitole jsou popsány všechny důležité úkony, které je nutné provést k navržení datového modelu tak, aby s ním bylo co možno nejlépe pracovat při navrhování aplikačního rozhraní.

Princip nového aplikačního rozhraní je až na pár detailů stejný, jako původní, nicméně pár rozdílných věcí se zde přesto najde. Pokud chce uživatel zadávat údaje o studentech, musí přejít přes přihlašovací okno do samotné aplikace, což je první rozdíl mezi starým a novým řešením. Toto je velice důležitý prvek z toho důvodu, že se jedná o webovou aplikaci, tím pádem by se do ní měli dostat pouze oprávnění uživatelé. Po úspěšném přihlášení je uživatel přesměrován na domovskou stránku, ze které se pomocí menu může dostat k vkládání nového záznamu. Uživatel vloží do systému údaje o studentovi, o jeho výjezdu a zkoušce. Logicky se do systému napřed zadá přihlášený student na zkoušku a poté, až ji vykoná, jsou k němu přidány údaje o výjezdu. Posléze lze vytisknout sestavy, podle potřeb a požadavků uživatele. Obrázek 4.3 znázorňuje sekvenci činností, jež byly popsány v předchozím odstavci pomocí diagramu aktivit tak, jak jsou v novém aplikačním rozhraní vytvořeny.



Obrázek 4.3 Activity diagram evidenčního systému (zdroj: vlastní)

Nový systém z uživatelského hlediska funguje obdobně, jako staré řešení, nicméně přibyly akce přihlášení a odhlášení ze systému. Znovu zde figuruje pouze uživatel obsluhující aplikaci, ani v tomto řešení není nutno přidávat žádného nového účastníka (například administrátora apod.), což je pomocí nového případu užití na obrázku 4.4 nastíněno.



Obrázek 4.4 Případ užití webové aplikace (zdroj: vlastní)

Jak již bylo zmíněno, celá webová aplikace je implementována v jazyku Java, konkrétně pomocí technologie JavaServer Faces. Tato podkapitola se zabývá podrobněji implementovanými prvky a technologickými postupy, které byly při vytváření použity. K samotné implementaci je využit NetBeans IDE 8.0.2 s JDK 1.8 a server Glassfish 4.1 s potřebnými drivery pro připojení k SQL Serveru.

4.3.1 Class Diagram

Protože diagram tříd webové aplikace je docela rozsáhlý a je zbytečné jej dávat celý do práce, nachází se v příloze č. 1. Diagramy tříd jsou pro přehlednost rozděleny do částí, kde každý jeden diagram znázorňuje danou entitu, její controller, facade, logicky přidružené třídy a balíky.

4.3.2 Maven project

Pro vývoj aplikace je v prostředí NetBeans použit Maven project. Jedná se o nástroj pro správu, řízení a automatizaci buildů aplikací. Maven lze použít pro většinu programovacích jazyků, nicméně je podporován převážně jazykem Java. Cílem Mavenu je tedy usnadnit jak samotný build unifikace, tak i unifikovat samotný systém buildování, poskytovat kvalitní informace o projektu a také poskytnout transparentní přidávání nových funkcí. Maven projekty jsou řízeny a konfigurovány pomocí Project Object Model, který je uložen v souboru pom.xml. Jedná se o jednoduchou XML strukturu definující jednotlivé části projektu a jeho závislosti na externích knihovnách a nástrojích. XML soubor dále umožňuje definovat konstanty, které mohou využívat jednotlivé pluginy, nicméně tuto funkci není nutno v práci používat. Každý projekt má vlastní pom.xml soubor, čili pokud je projekt složen z více dílčích projektů, pom.xml dědí vlastnosti od nadřazených souborů a mohou přidávat další položky. Tato struktura umožňuje sestavit projekt jediným příkazem. Řešená webová aplikace se skládá pouze z jednoho projektu, proto je zde pouze jeden XML soubor. Jeho grafická podoba je znázorněna v příloze č. 2, kde je možno pozorovat stromovou strukturu pom.xml. Skládá se ze snapshotu aplikace a knihoven, které jsou v projektu použity.

Pro příklad je znázorněna část souboru pom.xml níže. Nesporná výhoda nastává v importu knihoven, kdy programátor nemusí ručně importovat vlastní stažené knihovny, ale stačí pouze napsat XML kód (tag dependency), nebo najít knihovnu v Maven repository a kód se sám vygeneruje.

Menší komplikace nastává v importování vlastních knihoven, které se musí už řešit složitěji, nicméně v této práci nejsou použity žádné vlastní knihovny, tudíž se netřeba touto problematikou zabývat, navíc databáze knihoven Maven je tak rozsáhlá, že možnost nenalezení nějaké existující knihovny potřebné pro tuto aplikaci, je minimální.

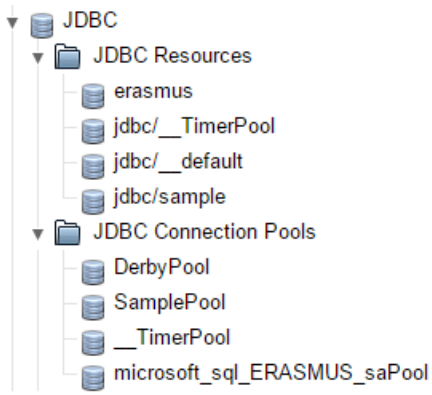
```
<groupId>com.mycompany</groupId>
<artifactId>Erasmus</artifactId>
<version>1.0-SNAPSHOT</version>
<packaging>war</packaging>

    <name>Erasmus</name>

    <properties>
        <endorsed.dir>${project.build.directory}/endorsed</endorsed.dir>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.eclipse.persistence</groupId>
            <artifactId>eclipselink</artifactId>
            <version>2.5.2</version>
            <scope>provided</scope>
        </dependency>
    </dependencies>
```

Příklad 4.8 Struktura souboru pom.xml

4.3.3 Připojení k databázi

Aby bylo možno pracovat s databází je nutno novému projektu zabezpečit přístup k databázi. To zabezpečuje jdbc driver, díky němuž získává NetBeans připojení k databázi a může pracovat s jejími objekty. K připojení je využit driver sqljdbc4.jar, volně přístupný na internetu. Toto připojení je použito při vyvíjení aplikace v prostředí NetBeans.

Obsluhu databáze, přístup jednotlivých vláken a připojení si už obsluhuje sám aplikační server Glassfish, kde se po deploymentu aplikace na server vytváří samotné připojení aplikace k databázi a pool vláken, viz obrázek 4.5. Poté je toto připojení aktuální tak dlouho, dokud aplikace není znovu deploynuta.



Obrázek 4.5 Připojení k aplikaci v administrátorském okně glassfish serveru (zdroj: vlastní)

4.3.4 Aplikační vrstva

Jak je zmíněno v kapitole o datovém modelování, dobře navržená databáze a její objekty výrazně usnadňují práci v aplikační vrstvě při vývoji aplikace. Aplikační vrstva se tedy zabývá jádrem aplikace, její logikou, výpočty a zpracováním dat.

Samotná aplikace je tvořena pomocí JavaServer Faces za použití frameworku PrimeFaces, který je použit na tvorbu prezentační vrstvy. Ukládání objektů do databáze a objektově relační mapovaní na aplikační vrstvě, zajišťuje Java Persistence Api (dále JPA).

Java Persistent Unit je logické uskupení uživatelem definovaných persistable tříd (entity classes, embeddable classes and mapped superclasses) se stejným nastavením. Definování Persistent Unit je povinná při použití JPA, nicméně například u ObjectDB není nutno persistentní jednotku deklarovat. Tyto jednotky jsou definovány v souboru persistence.xml, který je umístěn v META-INF složce. V jednom persistence souboru může být definováno jedna, nebo více persistentních jednotek, nicméně v této práci je definována pouze jedna. Tato jednotka je konfigurována v <persistence-unit> XML elementu, kde je povinný atribut name a transaction-type, který může nabývat hodnot JTA, nebo RESOURCE_LOCAL. Aplikace využívá JTA transaction manager, jelikož běží na Glassfish serveru a J2EE prostředí. Při nespecifikování příslušného transaction managera Glassfish generuje chybu. Dále je zde specifikována zdrojová databáze v elementu <jta-data-source> a obecné vlastnosti jednotky v <properties>. Struktura persistence.xml je zobrazena na příkladu 4.9.

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1" xmlns="http://xmlns.jcp.org/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">

  <persistence-unit name="com.mycompany_Erasmus_war_1.0-SNAPSHOTPU" transaction-type="JTA">
    <jta-data-source>erasmus</jta-data-source>
    <exclude-unlisted-classes>false</exclude-unlisted-classes>
    <properties>
      <property name="javax.persistence.schema-generation.database.action" value="create"/>
    </properties>
  </persistence-unit>

</persistence>
```

Příklad 4.9 Struktura persistence.xml

Ke každému objektu databáze, jenž je v tomto případě tabulka a pohled, je vytvořena „třída“, neboli entita. Jak již bylo psáno v předchozích kapitolách, jedná se v podstatě o objekt reprezentující data v databázi. Její instance reprezentuje jeden řádek tabulky. Každá entita musí mít vlastní identifikátor (@Id), který nesmí být null, což vyjadřuje anotace @NotNull. Bez identifikátoru nelze entitu vytvořit. Pro ukázku je vybrána entita tabulky department prezentující číselník kateder na EkF. U anotace @NamedQuery je zřejmé, že JPA nepoužívá standartní anotaci SQL, ale vlastní JPQL, jež je klasickému jazyku SQL velice podobné. Pro každý sloupec objektu databáze je definována proměnná s příslušným datovým typem odpovídajícím datovému typu v databázi, jeho názvu a velikosti. JPA samo zabezpečuje propojení entit a zajišťuje relace mezi nimi díky @JoinColumn a @OneToMany anotacím. Tyto multiplicity se mohou vyskytovat ve 4 modifikacích:

- **One-to-one:** instance má referenci na jednu entitu jiné třídy,
- **One-to-many:** instance má reference na množinou instancí jiné třídy,
- **Many-to-one:** více instancí má referenci na jednu instanci jiné třídy,
- **Many-to-many:** více instancí má reference na instance jiné třídy.

```
@Entity
@Table(name = "department")
@XmlRootElement
@NamedQueries({
        @NamedQuery(name = "Department.findAll", query = "SELECT d FROM Department d"),
        @NamedQuery(name = "Department.findByNumberDepartment", query = "SELECT d FROM Department d
        WHEREd.numberDepartment = :numberDepartment"),
        @NamedQuery(name = "Department.findByNameDepartment", query = "SELECT d FROM Department d WHERE
        d.nameDepartment = :nameDepartment")})
public class Department implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id
    @Basic(optional = false)
    @NotNull
    @Size(min = 1, max = 10)
    @Column(name = "number_department")
    private String numberDepartment;
    @Size(max = 100)
    @Column(name = "name_department")
    private String nameDepartment;
    @OneToMany(mappedBy = "numberDepartment")
    private Collection<Student> studentCollection;
/*gettery a settery*/ ...
}
```

Příklad 4.10 Ukázka struktury entity

Jak již bylo řečeno, každá entita má vlastní identifikátor, značený pomocí @Id, který musí být jedinečný. Menší komplikace nastává, když tento identifikátor nevyplňuje v prezentační vrstvě sám uživatel, ale jedná se pouze o nějakou automaticky generovanou hodnotu. Java Persistence Api má řešení i pro tento problém, které se ovšem liší v každé databázi (MySQL, Oracle, MS SQL Server). Je totiž třeba zabezpečit, že aplikační vrstva se nebude starat o zachování integrity databáze, ale že tento úkon obsluhuje samotná datová vrstva pomocí generování hodnoty při vkládání řádku a entita bude o tomto úkonu informována. Pokud je databáze vytvořená v prostředí MS SQL Server, je důležité v entitní třídě vložit do deklarace sloupce anotaci @GeneratedValue(strategy = GenerationType.IDENTITY). Tato anotace nefunguje na ostatních databázových platformách, kde se tento případ řeší trochu jinak, nicméně princip zůstává i tak zachován a jediný rozdíl je pouze v syntaxi. @GeneratedValue provádí inkrementaci hodnoty a GenerationType udává, že je nutno přiřadit primární klíč k entitě pomocí identity sloupce v databázi. Takto ošetřené vkládání primárního klíče oprošťuje uživatele od povinnosti vkládat ručně určitou sekvenci do sloupce identifikujícího tabulku.

4.3.4.1 *JPA Controller*

Ke každé entitě je vytvořen také příslušný Controller a Facade. JPA Controller obaluje entitu a poskytuje klientovi přístup do databáze pomocí metod v entitě. Obsahuje logiku pro vytváření, editování a mazání záznamů ve zdroji, selektování všech záznamů v databázi nebo selektování specifického záznamu. Pro ukázku jsou vybrány z DepartmentController metody pro create, update a delete z databáze.

```
public void create() {
      persist(PersistAction.CREATE,ResourceBundle.getBundle("/Bundle").getString("DepartmentCreated"));
      if (!JsfUtil.isValidationFailed()) {
         items = null;    // Invalidate list of items to trigger re-query.
        }
    }

    public void update() {
    persist(PersistAction.UPDATE,ResourceBundle.getBundle("/Bundle").getString("DepartmentUpdated"));
    }

    public void destroy() {
    persist(PersistAction.DELETE,ResourceBundle.getBundle("/Bundle").getString("DepartmentDeleted"));
        if (!JsfUtil.isValidationFailed()) {
            selected = null; // Remove selection
            items = null;    // Invalidate list of items to trigger re-query.
        }
    }
```

Příklad 4.11 Vybrané důležité metody z Controlleru

4.3.4.2 *JPA Facade*

Ke každé entitě je taktéž vytvořena příslušná třída Facade, která je stateless a rozšiřuje abstraktní třídu AbstractFacade, která obsahuje business logiku a spravuje transakce, jako vytváření, mazání a vyhledávaní entit. Pro ukázku slouží kód DepartmentFacade.

```
@Stateless
public class DepartmentFacade extends AbstractFacade<Department> {
    @PersistenceContext(unitName = "com.mycompany_Erasmus_war_1.0-SNAPSHOTPU")
    private EntityManager em;
    @Override
    protected EntityManager getEntityManager() {
        return em;
    }
    public DepartmentFacade() {
        super(Department.class);
    }
}
```

Příklad 4.12 Ukázka kódu JPA Facade

4.3.4.3 *Login*

Aby byla webová aplikace nějak zabezpečena proti nežádoucímu vniknutí, je třeba vytvořit určitý přihlašovací systém. Princip je podobný jako u předchozích vytvořených entit. I pro login je vytvořena v databázi tabulka s atributy pro jméno a heslo a nad ní je v aplikační vrstvě vytvořena entita a controller. Entita vypadá obdobně jako ostatní, čili proměnné reprezentující atributy v databázi a jejich gettery a settery. V controlleru je logika následující:

```
@ManagedBean (name="login")
@SessionScoped
public class LoginController implements Serializable {

    private String username;
    private String password;
    private final DataQuery query = new DataQuery();

public String loginControl(){
        if(query.loginControl(username, password)){
            return "index.xhtml?faces-redirect=true";
        }else{
        RequestContext.getCurrentInstance().update("growl");
        FacesContext context = FacesContext.getCurrentInstance();
        context.addMessage(null,new FacesMessage(FacesMessage.SEVERITY_ERROR, "Error", "Nesprávné
        uživatelské jméno nebo heslo."));
        }
        return "";
    }
    public String getUsername() {
        return username;
    }
    public void setUsername(String username) {
        this.username = username;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
}
```

Příklad 4.13 Struktura přihlašovací logiky

Podmínka porovnává login a heslo z databáze se zadanými údaji v přihlašovací stránce pomocí instance třídy DataQuery. Pokud zjistí, že se údaje rovnají, pomocí return klauzule a hodnotě „index.xhtml?faces-redirect=true“ uživatele přesměruje na index.xhtml. Jestli zjistí, že se přihlašovací údaje neshodují, tak v else části podmínky zobrazí hlášení (growl) o nesprávnosti přihlašovacích údajů (context.addMessage). Metoda loginControl porovnává zadané hodnoty, s hodnotami databáze, pokud se shodují, vrátí true, jinak false.

```
public class DataQuery {
    EntityManagerFactory emf;
    EntityManager em;
    public DataQuery() {
        emf = Persistence.createEntityManagerFactory("com.mycompany_Erasmus_war_1.0-SNAPSHOTPU");
        em = emf.createEntityManager();
    }
    public boolean loginControl(String username, String password ){
        try {
            Login l = em.createNamedQuery("Login.control", Login.class).setParameter("username",
            username).setParameter("password", password).getSingleResult();
            if(l!=null){
                return true;
            }
            return false;
        } catch (Exception e) {
        return false;
        }
    }
}
```

Příklad 4.14 Třída DataQuery a její metoda loginControl

4.3.5 Prezentační vrstva

JavaServer Faces technologie má nespornou výhodu v tom, že odděluje prezentační a aplikační vrstvu, tudíž tato vrstva zobrazuje pouze informace pro uživatele pomocí grafického rozhraní, kontroluje zadávané vstupy, nicméně zpracování dat už nechává na aplikační vrstvě, jež je popsána v předchozí podkapitole.

Na tvorbu prezentační vrstvy existuje celá řada frameworků, které usnadňují jeho tvorbu, zejména pak IceFaces, RichFaces a PrimeFaces. V této práci je využit právě framework PrimeFaces verze 4.0, jenž byl zmíněn v kapitole 2.6.1.

4.3.5.1 *Layout*

Každá webová stránka musí mít určitý layout, který je neměnný a dynamicky zobrazuje prvky, které uživatel požaduje. V tomto případě je layout navržen ve stránce template.xhtml pomocí PrimeFaces anotací.

V aplikaci je využíván Ajax, zejména pak v menu, kde je v anotaci <p:menuitem> využíván atribut ajax, který je nastaven na true. Takhle si PrimeFaces jednoduše a automaticky zabezpečují ajax načítání stránek. Ajax je také použit v updatu tabulek po nějaké operaci (insert, delete, update).

```
<h:body>
  <p:growl id="growl" life="5000" />
  <p:layout fullPage="true">
   <p:layoutUnit position="north" size="60" header="International Office">
   </p:layoutUnit>
   <p:layoutUnit position="west" size="167" header="Menu">
    <h:form id="menuForm">
     <p:menu>
       <p:menuitem value="#{bundle.Home}" outcome="/index" icon="ui-icon-home" ajax="true"/>
         <p:submenu label="Editace">
            <p:menuitem value="Studenti" outcome="/student/List.xhtml" ajax="true"/>
            <p:menuitem value="Zkoušky" outcome="/studentExam/List.xhtml" ajax="true"/>
            <p:menuitem value="Katedry" outcome="/department/List.xhtml" ajax="true"/>
            <p:menuitem value="Univerzity" outcome="/university/List.xhtml" ajax="true"/>
         </p:submenu>
         <p:submenu label="Vložení záznamu">
            <p:menuitem value="Nový záznam" outcome="/student/ListOverview.xhtml" ajax="true"/>
         </p:submenu>
         <p:submenu label="Sestavy">
            <p:menuitem value="Přehled výjezdů" outcome="/exchangeView/List.xhtml" ajax="true"/>
            <p:menuitem value="Přehled zkoušek" outcome="/studentExamView/List.xhtml" ajax="true"/>
            <p:menuitem value="Celkový přehled" outcome="/overviewView/List.xhtml" ajax="true"/>
         </p:submenu>
     </p:menu>
    </h:form>
   </p:layoutUnit>
   <p:layoutUnit position="south" size="25">
       <ui:insert name="footer"/>
   </p:layoutUnit>
   <p:layoutUnit position="center">
       <ui:insert name="body"/>
   </p:layoutUnit>
  </p:layout>
</h:body>
```

Příklad 4.15 Struktura template.xhtml a konfigurace layoutu

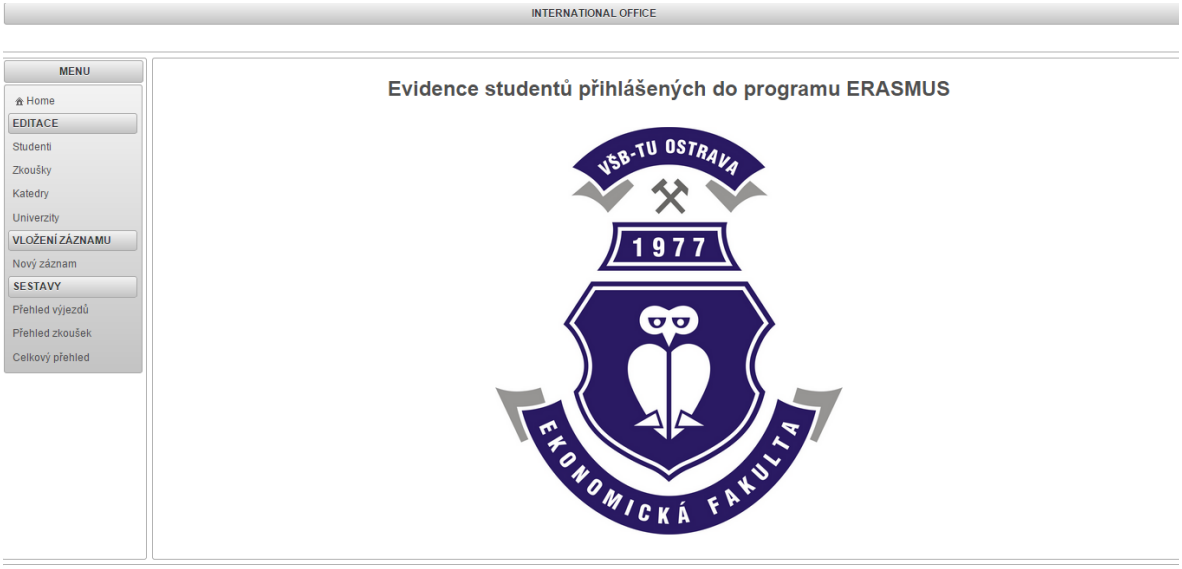
Díky <p:layout> a vnořených <p:layoutUnit> je navrženo základní rozložení headeru, footeru, hlavního panelu a menu. Menu je navrženo pomocí <p:menu>, <p:menuitem> a <p:submenu>. Anotace <p:growl> se svým atributem life udává, jak dlouho trvá zobrazení informační zprávy v milisekundách. Například při chybném vložení záznamu, špatném zadání hesla apod.

V každé stránce je pak definována template.xhtml jako defaultní layout, který je používán napříč celou aplikací. Například index.xhtml vypadá takto:

```
<h:body>
 <ui:composition template="template.xhtml">
  <ui:define name="title">
      <h:outputText value="Hlavní stránka"></h:outputText>
  </ui:define>
  <ui:define name="body">
   <h:form id="welcome">
        <H1 style="text-align:center;">Evidence studentů přihlášených do programu ERASMUS</H1><br/>
        <img src="resources/images/LOGO_EKF_CS.jpg" alt="ekf"
        style="width:401.25px;height:470px;display:block;margin-left:auto;margin-right:auto;"/>
   </h:form>
  </ui:define>
 </ui:composition>
</h:body>
```

Příklad 4.16 Práce s template.xhtml v ostatních stránkách

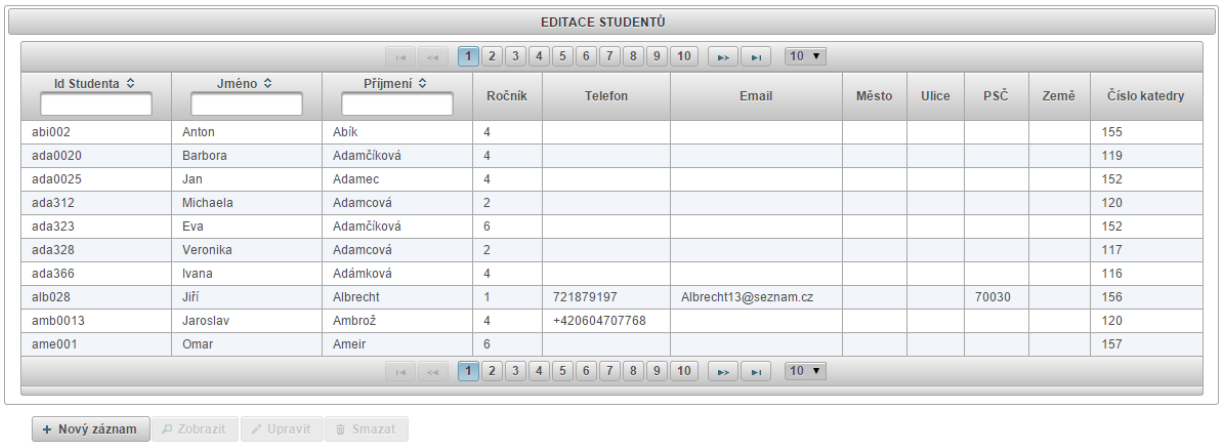
Pomocí anotace <ui:composition> je definována template layoutu, kde <ui:define> určuje rozložení, jak vypadá tělo stránky. Obrázek 4.7 znázorňuje základní rozložení stránky, kdy se jedná konkrétně o index.xhtml.



Obrázek 4.6 Layout stránek

4.3.5.2 *JSF stránky z entit*

Nad každou entitou je vytvořena sada xhtml stránek. První stránkou je List.xhtml, který zobrazuje základní tabulku s daty (viz obrázek 4.8).



Obrázek 4.7 Přehled studentů

Tato stránka je tvořena jednou tabulkou pomocí <p:dataTable> s důležitými atributy, které definují tabulku tak, jak je ji možno vidět právě na předchozím obrázku.

```
<p:dataTable id="datalist" value="#{studentController.items}" var="item"
                                selectionMode="single" selection="#{studentController.selected}"
                                paginator="true"
                                rowKey="#{item.shortcutidStudent}"
                                rows="10"
                                rowsPerPageTemplate="10,20,30,40,50"
                                >
             <p:ajax event="rowSelect"   update="createButton viewButton editButton deleteButton"/>
             <p:ajax event="rowUnselect" update="createButton viewButton editButton deleteButton"/>
...
</p:dataTable>
```

Příklad 4.17 Definice atributů v tabulce

Nejdůležitější atributy dataTable jsou:

- id – určuje název tabulky, se kterým se pracuje a slouží ke komunikaci aplikační vrstvy s prezentační,
- value – díky této anotaci se do tabulky plní data z příslušné entity,
- paginator – definuje, zda má mít tabulka více stránek,
- rowKey – jednoznačně definuje řádek, v tomto případě se jedná o id studenta,
- rows – udává počet řádků na jednu stránku,
- rowsPerPageTemplate – díky tomuto atributu lze navolit, kolik řádků se bude zobrazovat v tabulce.

Tabulka se samozřejmě skládá ze sloupců, které jsou definovány pomocí anotace <p:column>, jehož struktura je zobrazena níže. Pro ilustraci je vybrán sloupec s id studenta, který zachycuje většinu použitých atributů.

```
<p:column sortBy="#{item.shortcutidStudent}" filterBy="#{item.shortcutidStudent}">
                            <f:facet name="header">
                                <h:outputText value="Id Studenta"/>
                            </f:facet>
                                <h:outputText value="#{item.shortcutidStudent}"/>
</p:column>
```

Příklad 4.18 Deklarace sloupce v tabulce

Atributy sortBy a filterBy slouží k seřazování a filtrování dat podle sloupce, vše funguje s ajaxem, čili není třeba danou činnost potvrzovat tlačítkem, všechny úkony jsou vykonány okamžitě (viz podkapitola Ajax). Velice důležité jsou vnořené stránky, které zabezpečují vytváření záznamu, úpravu záznamu a náhled záznamu. Tyto stránky, jejichž názvy jsou Edit.xhtml, Create.xhtml a View.xhtml, jsou zahrnuty pomocí <ui:include> a atributu src.

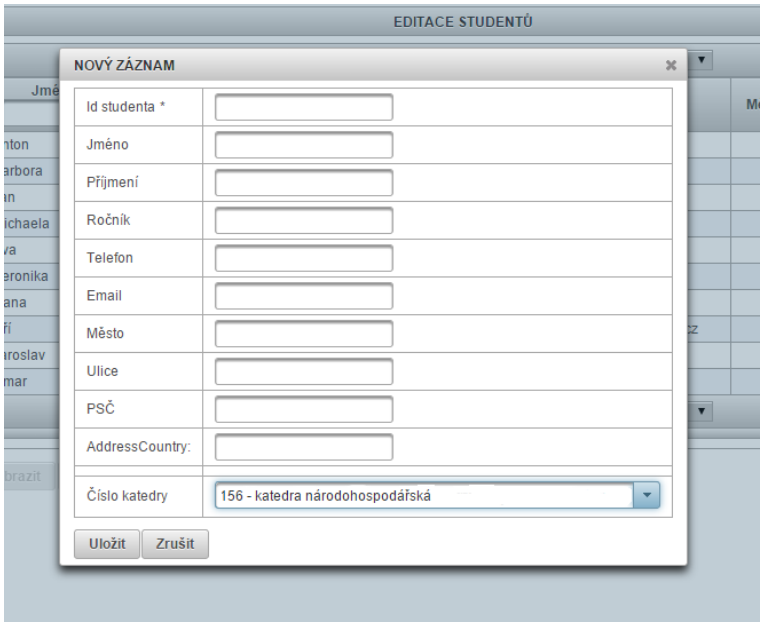
```
<ui:include src="Create.xhtml"/>
<ui:include src="Edit.xhtml"/>
<ui:include src="View.xhtml"/>
```

Příklad 4.19 Vnoření stránek

Tyto stránky jsou volány tlačítky „Nový záznam“, „Upravit“, „Zobrazit“ a „Vymazat“. Vyjma tlačítka pro vytvoření nového záznamu jsou všechna další neaktivní a aktivují se až po označení řádku v tabulce, se kterým chce uživatel provést určitou operaci.

Tlačítko na mazání záznamů nevolá žádnou stránku, ale pouze vymaže určitý řádek. Samozřejmě, pokud by touto operací byla porušena integrita databáze, aplikace uživateli vymazat takovýto řádek nedovolí. Stejně tak je zabezpečena integrita při vkládání nových záznamů a upravování.

Stránka „Nový záznam“ funguje jako lightbox, který se zobrazí nad hlavním layoutem, kde uživatel vyplňuje všechny záznamy, které potřebuje, viz obrázek 4.9, kde je uveden příklad s vložením nového studenta. Atributy s hvězdičkou jsou povinné atributy, bez jejichž vyplnění uživatel nový záznam nevloží. Jedná se převážně o primární klíče. Takhle prezentační vrstva předchází prvotním problémům s porušováním integrity databáze. Další možnou chybou je, že uživatel vloží do textového pole, kde jsou vyžadována pouze čísla nějaké jiné znaky, i v tomto případě jej aplikace nepustí a transakci přeruší. Zadávání cizího klíče je vyřešeno pomocí roletky se seznamem cizích klíčů, aby uživatel nemohl zadat nějakou neexistující hodnotu a nedošlo tak k vyhození chyby databáze. Mezi klíči lze vyhledávat, čili odpadá nepříjemné scrollování záznamy.



Obrázek 4.8 Nový záznam do databáze

Lightbox je vytvořen pomocí anotace <p:dialog> který má určité atributy, zajišťující jeho formátování (viz kód níže) v němž je definováno pole o dvou sloupcích, ve kterých jsou zobrazeny labely a textová pole, do kterých uživatel vkládá záznam.

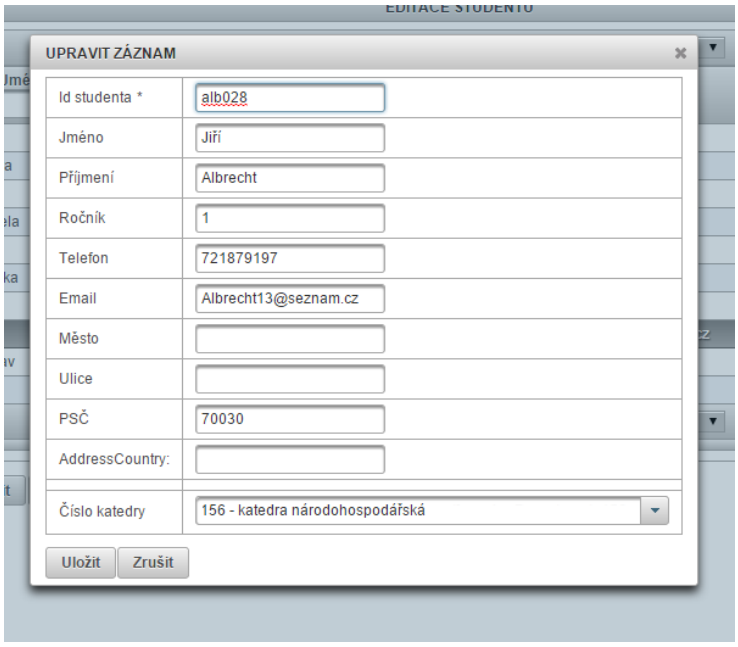
Pro úsporu místa je zde kód pouze pro první atribut id studenta, label je vytvořen pomocí <p:outputLabel> a <p:inputText> je textové pole, kde pomocí atributu value předává aplikační vrstvě údaje, které je třeba vložit do databáze. Atribut title je název textového pole. Pokud není toto pole vyplněno, zobrazí se upozornění, že je třeba dané pole vyplnit, viz atribut requiredMessage.

Pro potvrzení vložení záznamu jsou zde tlačítka Uložit a zavření dialogového okna tlačítko Zrušit. Tlačítko Uložit po stisku uloží data do databáze pomocí instance controlleru a updatuje tabulku na stránce. Tlačítko Zrušit nedělá nic jiného, než že zavře dialogové okno.

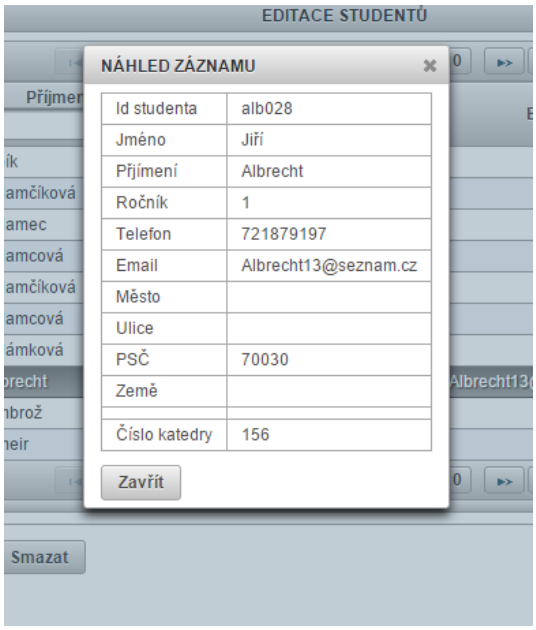
```
<p:dialog id="StudentCreateDlg" widgetVar="StudentCreateDialog" modal="true" resizable="false"
appendTo="@(body)" header="Nový záznam">
 <h:form id="StudentCreateForm">
  <h:panelGroup id="display">
   <p:panelGrid columns="2" rendered="#{studentController.selected != null}">
    <p:outputLabel value="Id studenta" for="shortcutidStudent" />
        <p:inputText id="shortcutidStudent" value="#{studentController.selected.shortcutidStudent}"
        title="#{bundle.CreateStudentTitle_shortcutidStudent}" required="true"
        requiredMessage="#{bundle.CreateStudentRequiredMessage_shortcutidStudent}"/>
...
   </p:panelGrid>
        <p:commandButton actionListener="#{studentController.create}" value="Uložit"
        update="display,:StudentListForm:datalist,:growl"
        oncomplete="handleSubmit(args,'StudentCreateDialog');"/>
        <p:commandButton value="Zrušit" onclick="StudentCreateDialog.hide()"/>
  </h:panelGroup>
 </h:form>
</p:dialog>
```

Příklad 4.20 Kód dialogového okna na vytvoření záznamu

Obdobně jako vytvoření nového záznamu funguje také úprava záznamu a náhled záznamu ale s tím rozdílem, že je třeba označit v tabulce dotyčný řádek a až poté se aktivují tlačítka na vyvolání těchto akcí. Kód těchto stránek vypadá obdobně jako u nového záznamu. Taktéž se jedná o lightbox, který má stejné formátování, ale jediný rozdíl je v prováděných akcích po stisku tlačítek, kdy u nového záznamu byla volána metoda create(), kdežto u úpravy záznamu je logicky volána metoda update(). U náhledu není editace vůbec povolena. Jak tyto stránky vypadají, lze pozorovat na obrázcích 4.10 a 4.11.



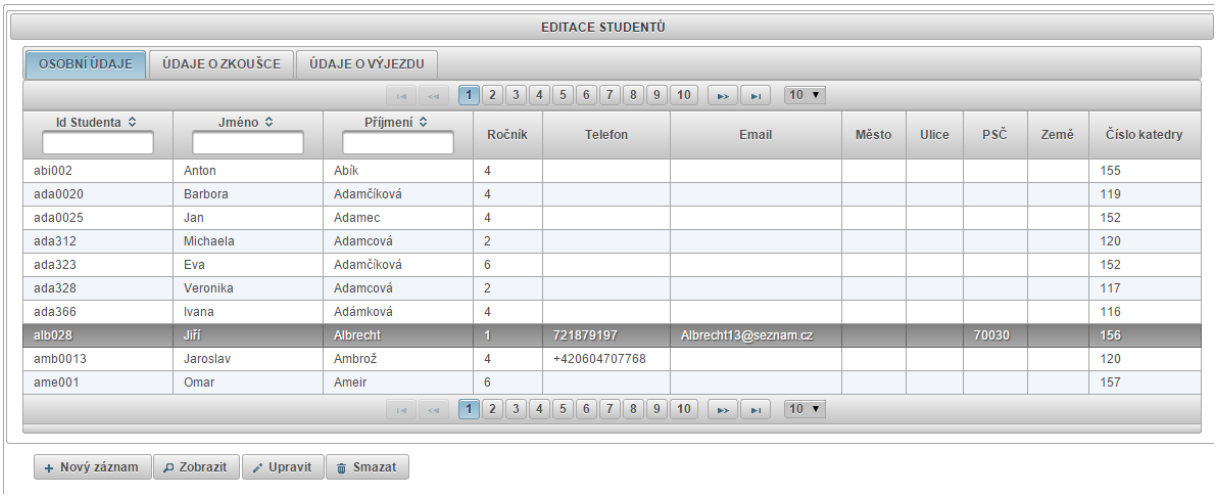
Obrázek 4.9 Dialogové okno úpravy záznamu



Obrázek 4.10 Dialogové okno náhledu záznamu

4.3.5.3 *Přidávání celkového záznamu*

Původní okno aplikace řešilo přidávání nového záznamu studenta, jeho zkoušky a výjezdu v jednom okně, které bylo složeno z více formulářů. Toto řešení sice bylo přehledné, nicméně stačilo nevyplnit jeden povinný údaj a aplikace vyhodila nesčetně mnoho chybových hlášek. Tomuto se snaží tato aplikace předejít tím, že rozděluje celý jeden insert, který byl v původním řešení, na tři samostatné úkony. Tímto se zabezpečí integrita dat, kdy se nemůže už stát, že se bude do databáze vkládat zkouška neexistujícímu studentovi a podobně.



Obrázek 4.11 Vložení celkového záznamu

Princip práce s daty je stejný, jako v předchozích případech, tudíž nemá smysl se o tom dále rozepisovat. Důležitým prvkem zde jsou záložky, které jsou vytvořeny pomocí anotace <tabMenu>. Tato značka má důležitý atribut activeIndex, který má hodnotu *i*, jež se předává z jednoho menuitem na druhý, což zabezpečuje, že aktivní záložka je zvýrazněna modrou barvou. Pokud je hodnota i rovna nule, tak je aktivní první záložka, pokud je rovna jedné, tak druhá záložka a hodnota parametru rovna dvěma zajišťuje aktivní třetí záložku. Tato operace je zajištěna díky anotaci <f:param> s atributy name a value, což ostatně lze vyčíst z níže přiložené ukázky kódu.

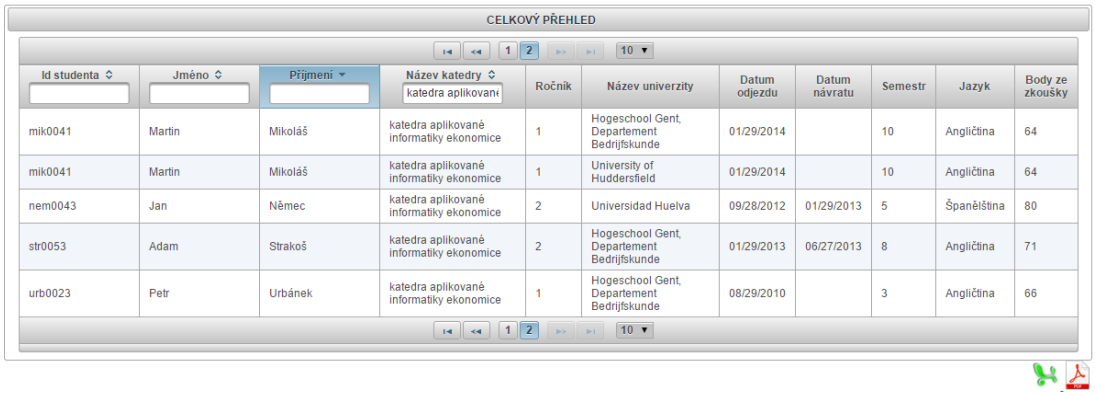
```
<p:tabMenu activeIndex="#{param.i}">
                        <p:menuitem value="Osobní údaje" outcome="/student/ListOverview.xhtml">
                            <f:param name="i" value="0" />
                        </p:menuitem>
                        <p:menuitem value="Údaje o zkoušce" outcome="/studentExam/ListOverview.xhtml">
                            <f:param name="i" value="1" />
                        </p:menuitem>
                        <p:menuitem value="Údaje o výjezdu" outcome="/exchange/ListOverview.xhtml">
                            <f:param name="i" value="2" />
                        </p:menuitem>
</p:tabMenu>
```

Příklad 4.21 Deklarace záložek

4.3.5.4 *Tisk sestav*

Jednou z funkcí původní aplikace byl tisk sestav. Tento tisk sestav nebyl příliš uživatelsky příjemný, neboť například pro tisk sestavy o zkouškách uživatel musel znát přesný datum zkoušky, musel datum zadat ve správném formátu a také tyto údaje byly case sensitive, čili „angličtina“ a „Angličtina“ byla rozdílná slova. Tato aplikace toto omezení do jisté míry odstraňuje.

Celkem je možno vytisknout tři sestavy, pro které byly dříve vytvořeny tři různé pohledy v databázi. První je celkový přehled, který je na obrázku 4.13, druhý pohled a následná sestava zobrazuje zkoušky studentů a poslední jejich výjezdy. Filtrování a seřazování, které bylo v Access aplikaci vyřešeno pomocí dialogového okna, se zde řeší přímo filtrem sloupce. Čili pokud uživatel chce studenty pouze z katedry aplikované informatiky (viz obrázek), tak jednoduše zadá do filtru název katedry. Jeho funkce byla popsána v předchozích kapitolách.


CELKOVÝ PŘEHLED

Id studenta	Jméno	Příjmení	Název katedry	Ročník	Název univerzity	Datum odjezdu	Datum návratu	Semestr	Jazyk	Body ze zkoušky
mik0041	Martin	Mikoláš	katedra aplikované informatiky ekonomice	1	Hogeschool Gent, Departement Bedrijfskunde	01/29/2014		10	Angličtina	64
mik0041	Martin	Mikoláš	katedra aplikované informatiky ekonomice	1	University of Huddersfield	01/29/2014		10	Angličtina	64
nem0043	Jan	Němec	katedra aplikované informatiky ekonomice	2	Universidad Huelva	09/28/2012	01/29/2013	5	Španělština	80
str0053	Adam	Strakoš	katedra aplikované informatiky ekonomice	2	Hogeschool Gent, Departement Bedrijfskunde	01/29/2013	06/27/2013	8	Angličtina	71
urb0023	Petr	Urbánek	katedra aplikované informatiky ekonomice	1	Hogeschool Gent, Departement Bedrijfskunde	08/29/2010		3	Angličtina	66

Obrázek 4.12 Tisk sestavy

Na rozdíl od tabulek, pohledy nejsou editovatelné, čili slouží opravdu jenom pro přehled záznamů.

Následující tisk je možný buď do formátu pdf, nebo excelu. Na rozdíl od exportu v klasických desktopových aplikacích je webový export podstatně jednodušší, jelikož funguje díky importovaným knihovnám a anotaci <p:dataExporter>, který díky atributu target zajistí export příslušné tabulky, atributu type export do požadovaného formátu a fileName, jenž pojmenuje exportovaný soubor.

Export souboru je aplikován na aktuální tabulku v aktuálním stavu. Čili pokud je v tabulce aplikován nějaký filtr, či řazení, exportují se pouze vyfiltrovaná či seřazená data.

```
<facet name="export" style="float:right;">
                <h:commandLink>
                    <img src="../resources/images/xls.png" alt=""/>
                    <p:dataExporter type="xls" target="datalist" fileName="Sestava vyjezdu" />
                </h:commandLink>
                <h:commandLink>
                    <img src="../resources/images/pdf.png" alt=""/>
                    <p:dataExporter type="pdf" target="datalist" fileName="Sestava vyjezdu"/>
                </h:commandLink>
</facet>
```

Příklad 4.22 PrimeFaces export tabulky do souboru

Na export do excelu a formátu pdf jsou použity tyto dvě knihovny dostupné z Maven repository.

```
<!--export excel-->
<dependency>
    <groupId>com.lapis.jsfexporter</groupId>
    <artifactId>export-type-excel</artifactId>
    <version>1.0.2.Final</version>
</dependency>
<!--export pdf-->
<dependency>
    <groupId>com.lowagie</groupId>
    <artifactId>itext</artifactId>
    <version>4.2.1</version>
</dependency>
```

Příklad 4.23 Import knihoven pro export v dependencies

4.3.5.5 *Bundle properties*

Obvykle při prvním vytváření aplikace se přehlíží nastavení zdrojových zpráv a popisných labelů. To ve výsledku přináší komplikaci v tom, že programátor při nutnosti změny určitého labelu musí ve zdrojovém kódu, přepisovat všechny názvy ručně. Tento problém už byl vyřešen ve verzi JSF 1.2 pomocí resource bundlů, které sice měly daleko k dokonalosti, nicméně od JSF 2.0 a novější je užívání bundlů velice jednoduché a pro vývoj nezbytné.

Bundle properties je soubor, kde jsou deklarovány zdroje zpráv a oznámení, používaných v aplikaci, stejně jako labely objektů a podobně. Bundle je deklarován pomocí value, což je deklarace proměnné a stringového řetězce, jenž je obalen value hodnotou. Zdrojové zprávy lze deklarovat i v kódu pomocí Java Beanů, getterů a setterů, nicméně tohoto řešení není v aplikaci využito. Jednoduše řečeno, bundle properties funguje na podobném principu jako css styly, kdy pokud bude chtít programátor změnit název tlačítka v celé aplikaci, nemusí měnit atribut value pro každé tlačítko zvlášť, ale stačí změnit stringovou hodnotu v bundle properties a změna bude viditelná ve všech objektech, které mají value nastavenou na stejnou bundle hodnotu, která byla změněna.

```
Cancel=Zrušit
Save=Uložit
SelectOneMessage=Vyber
Home=Domů
AppName=Erasmus
DepartmentCreated=Záznam katedry byl úspěšně přidán.
DepartmentUpdated= Záznam katedry byl úspěšně upraven.
```

Příklad 4.24 Deklarace bundle.properties

Tento soubor je třeba uložit v balíčku, který je na stejném místě jako zdrojový kód, nicméně Maven poskytuje oddělené místo pro tyto soubory, které je v src/main/resources. Nezbytným krokem je deklarovat ve faces-config.xml tento soubor pomocí následujícího kódu.

```
<resource-bundle>
    <base-name>/Bundle</base-name>
    <var>bundle</var>
</resource-bundle>
```

Příklad 4.25 Deklarace bundle.properties ve faces-config.xml

Volání bundlu v kódu je taktéž velice jednoduché a funguje obdobně, jako volání Java Beanů pomocí atributu value="#{bundle.EditOverviewViewTitle_shortcutidStudent}", kde se labelu předá stringová hodnota „Id studenta“.

4.4 Zhodnocení navržené koncepce a její implementace

Nejdůležitějším krokem realizace celého systému je důkladná analýza uživatelských požadavků, jež byly stanoveny konzultací s koncovými uživateli vyvíjeného systému. Tento systém je postaven na architektuře klient - server a třívrstvém modelu. Díky oddělení prezentační a aplikační logiky přináší nesmírné výhody v celkovém návrhu, kdy prezentační vrstva nemusí řešit složité procesy, jež obstarává aplikační vrstvy (manipulace s daty, perzistentní uchování dat a integritní omezení…), ale může se zabývat pouze zobrazováním výsledků a voláním právě aplikační logiky. Třetí vrstvu plní databázový systém.

5 Závěr

Cílem této práce bylo vytvořit informační systém pro International Office na Ekonomické fakultě VŠB – TUO. Tento systém měl za úkol usnadnit evidenci a přehled studentů přihlášených na program Erasmus, jejich výjezdů a zkoušek. Z tohoto důvodu je systém koncipován tak, že je možný přístup do něj odkudkoliv a kdykoliv, čili jako webová aplikace propojena s databází.

Metodická část se snažila přiblížit a vysvětlit problematiku modelování informačních systémů a jejich vývoje. Nedílnou součástí je důkladný popis použitých technologií, které byly k vývoji použity, kdy se jedná převážně o JavaServer Faces použité v návrhu a implementaci aplikační a prezentační vrstvy a využití MS SQL Server 2012 a jeho doplňku při návrhu datové vrstvy.

Následující kapitola shrnula aktuální stav současného evidenčního systému, jeho nedostatky a chyby, následované návrhem koncepce nového řešení v podobě webové aplikace. Samotná implementace této koncepce byla popsána v další kapitole. Prvním krokem bylo důkladné navržení datové vrstvy v podobě konceptuálního modelu a následného převedení do modelu relačního, který byl aplikován v prostředí MS SQL Server 2012. Posléze byla provedena transformace dat z původní databáze do nově vytvořené. Následující krok měl za úkol namodelovat aplikační a prezentační vrstvu pomocí UML. Díky tomuto kroku už bylo snadné tyto vrstvy v prostředí NetBeans IDE 8.0.2 implementovat. Samotná implementace probíhala aprezentační vrstvy.

Aplikace po splnění všech kroků plní všechny funkce, které byly uživateli zadány. Všechny cíle, a to včetně průběžných, které vyvstaly během implementace nebo při konzultacích s koncovými uživateli, byly splněny. Systém byl otestován a je připraven pro uvedení do ostrého provozu. Je nutno vzít v potaz, že právě během každodenního užívání systému se mohou objevit nedostatky, které si mohou vyžádat určité úpravy. Další vývoj aplikace tedy bude záležet na požadavcích uživatelů, nicméně dá se říct, že už nyní je plně funkční a úpravy by měly být v současném rozsahu minimální.

Seznam použité literatury

Literatura

ARLOW, J., B. KISZKA a I. NEUSTADT. UML 2 a unifikovaný proces vývoje aplikací: objektově orientovaná analýza a návrh prakticky. Brno: Computer Press, 2007, 567 s. ISBN 978-80-251-1503-9.

DEWSON, Robin. *Beginning SQL Server 2012 for Developer, 3rd Edition.* New York: Apress, 2012. 697 s. ISBN 978-1-4302-3750-1.

FOWLER, Martin. *Destilované UML*. Vyd. 3. Praha : Grada Publishing, 2009. 176 s. ISBN 978-80-247-2062-3.

HEFFELFINGER, R. David. *Java EE 6 Development with Netbeans 7*. Birmingham: Packt Publishing Ltd., 2011. 374 s. ISBN 978-1-849512-70-1.

KADLEC, Václav. *Agilní programování : Metodiky efektivního vývoje softwaru*. Brno: Computer Press, 2004. 278 s. ISBN 80-251-0342-0.

KALUŽA, Jindřich a Ludmila KALUŽOVÁ. *Modelování dat v informačních systémech*. Praha: Ekopress, 2012. ISBN 978-80-86929-81-1.

MISTRY, Ross. *Introducing Microsoft SQL Server 2012*. Redmond, WA: Microsoft Press, 2012, p. cm. ISBN 9780735665156.

ŠIMONOVÁ, Stanislava a Jan PANUŠ. *Databázové systémy I.* Pardubice: Univerzita Pardubice, 2007. 106 s. ISBN 978-80-7194-988-6.

TONG, Kent Ka Iok. *Beginning JSF 2 APIs and JBoss Seam*. New ed. Berkeley, CA: Apress, 2009. ISBN 9781430219224.

TVRDÍKOVÁ, Milena. Aplikace moderních informačních technologií v řízení firmy. Praha: Grada Publishing, 2008. 978-80-247-2728-8.

Internetové zdroje

Developing with Java Persistence. *Oracle* [online]. 2013 [cit. 2015-03-28]. Dostupné z: http://docs.oracle.com/cd/E40938_01/doc.74/e40142/dev_persistence.htm#BABCGACD

International Office. *Ekonomická fakulta - VŠB - TUO* [online]. 2015 [cit. 2015-03-21]. Dostupné z:http://www.ekf.vsb.cz/k163/cs

NetBeans. *NetBeans IDE* [online]. 2011 [cit. 2015-03-20]. Dostupné z: https://netbeans.org/features/index.html

PrimeFaces. *PrimeFaces* [online]. 2014 [cit. 2015-03-20]. Dostupné z: http://www.primefaces.org/whyprimefaces

Testování softwaru [online]. 2013 [cit. 2015-02-04]. Dostupné z: http://testovanisoftwaru.cz/manualni-testovani/modely-zivotniho-cyklu-softwaru/vodopadovy-model/

The Java EE 5 Tutorial. Oracle [online]. 2007 [cit. 2015-04-07]. Dostupné z:http://docs.oracle.com/javaee/5/tutorial/doc/bnbqa.html#bnbqc

Seznam zkratek

1NF – První normální forma

2NF – Druhá normální forma

3NF – Třetí normální forma

AJAX - Asynchronous JavaScript

API – Application Programming Interface

BCNF – Boyce – Coddova normální forma

CASE – Computer-Aided Software Engineering

DBMS – Database Management System

DSS – Decision Support System

EIS – Execution Information System

ER – Entity-Relationship

FK – Foreign Key

HTML – HyperText Markup Language

IDE – Integrated Development Environment

JDK – Java Development Kit

JPA – Java Persistence Api

JPQL – Java Persistence Query Language

JSF – JavaServer Faces

JSP – JavaServer Pages

PDF – Portable Document Format

PHP – Hypertext Preprocessor

PK – Primary Key

SQL – Structure Query Language

TPS – Transaction Processing System

T-SQL – Transact Structure Query Language

UI – User Interface

UML – Unified Modelling Language

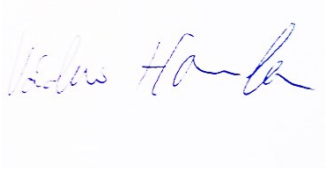
XML – Extensible Markup Language

Prohlášení o využití výsledků diplomové práce

Prohlašuji, že

- jsem byl seznámen s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. – autorský zákon, zejména § 35 – užití díla v rámci občanských a náboženských obřadů, v rámci školních představení a užití díla školního a § 60 – školní dílo;
- beru na vědomí, že Vysoká škola báňská – Technická univerzita Ostrava (dále jen VŠB-TUO) má právo nevýdělečně, ke své vnitřní potřebě, diplomovou práci užít (§ 35 odst. 3);
- souhlasím s tím, že diplomová práce bude v elektronické podobě archivována v Ústřední knihovně VŠB-TUO a jeden výtisk bude uložen u vedoucího diplomové práce. Souhlasím s tím, že bibliografické údaje o diplomové práci budou zveřejněny v informačním systému VŠB-TUO;
- bylo sjednáno, že s VŠB-TUO, v případě zájmu z její strany, uzavřu licenční smlouvu s oprávněním užít dílo v rozsahu § 12 odst. 4 autorského zákona;
- bylo sjednáno, že užít své dílo, diplomovou práci, nebo poskytnout licenci k jejímu využití mohu jen se souhlasem VŠB-TUO, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly VŠB-TUO na vytvoření díla vynaloženy (až do jejich skutečné výše).

V Ostravě dne 25. dubna 2015

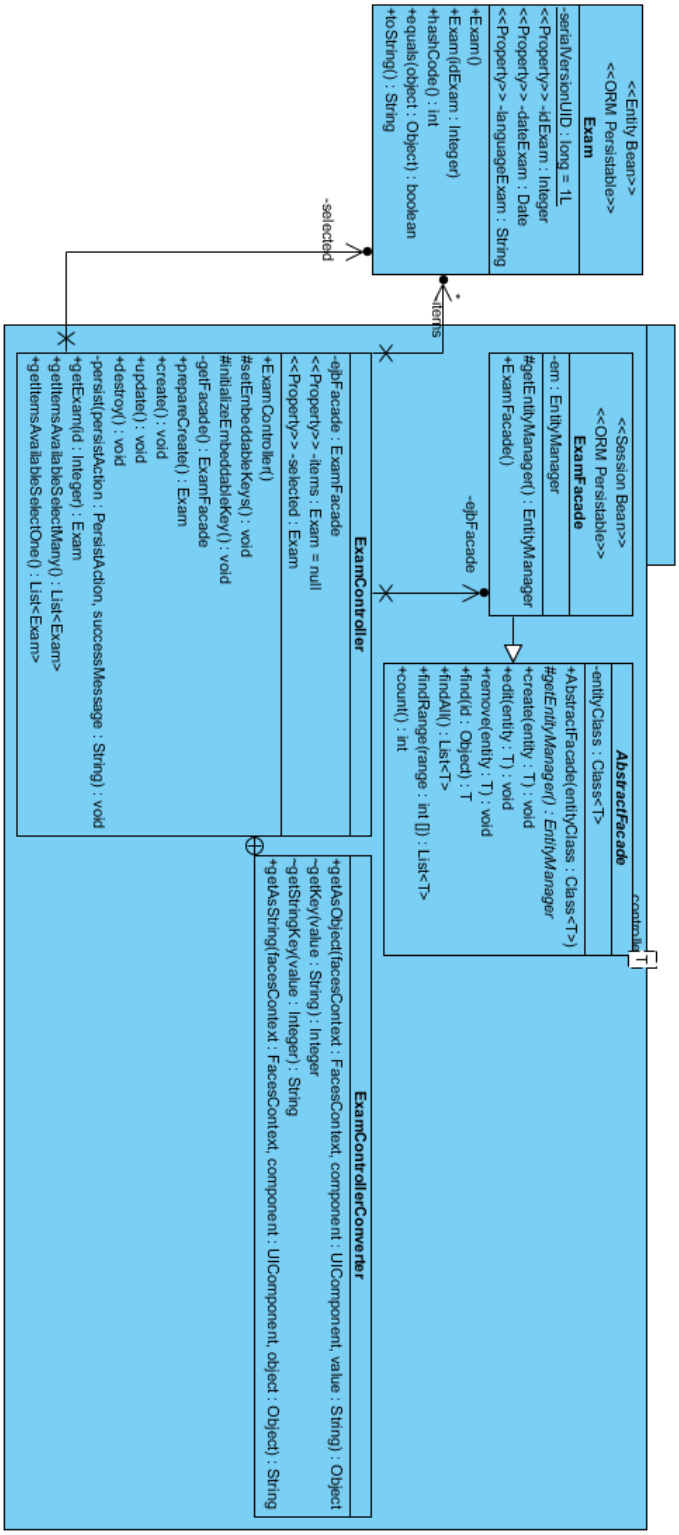


.....

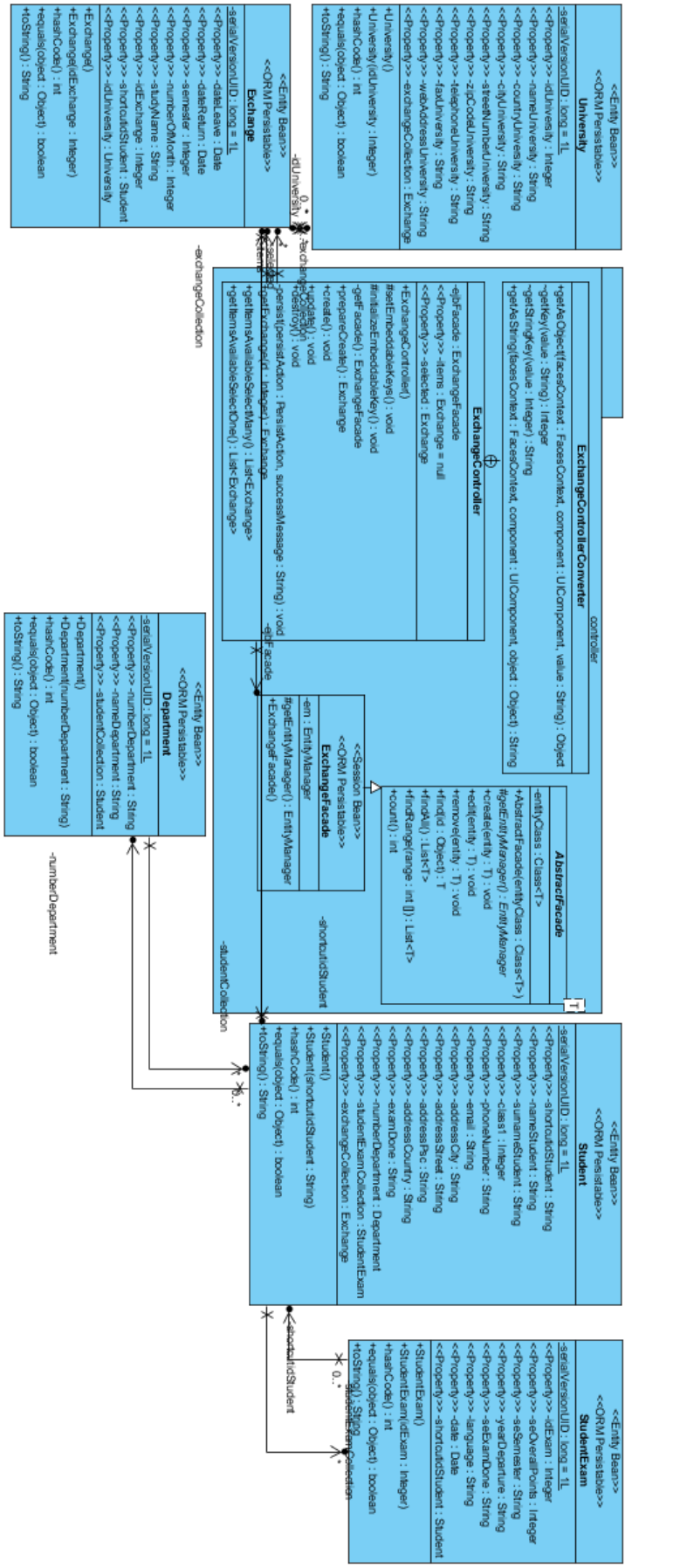
Václav Homola

Seznam příloh

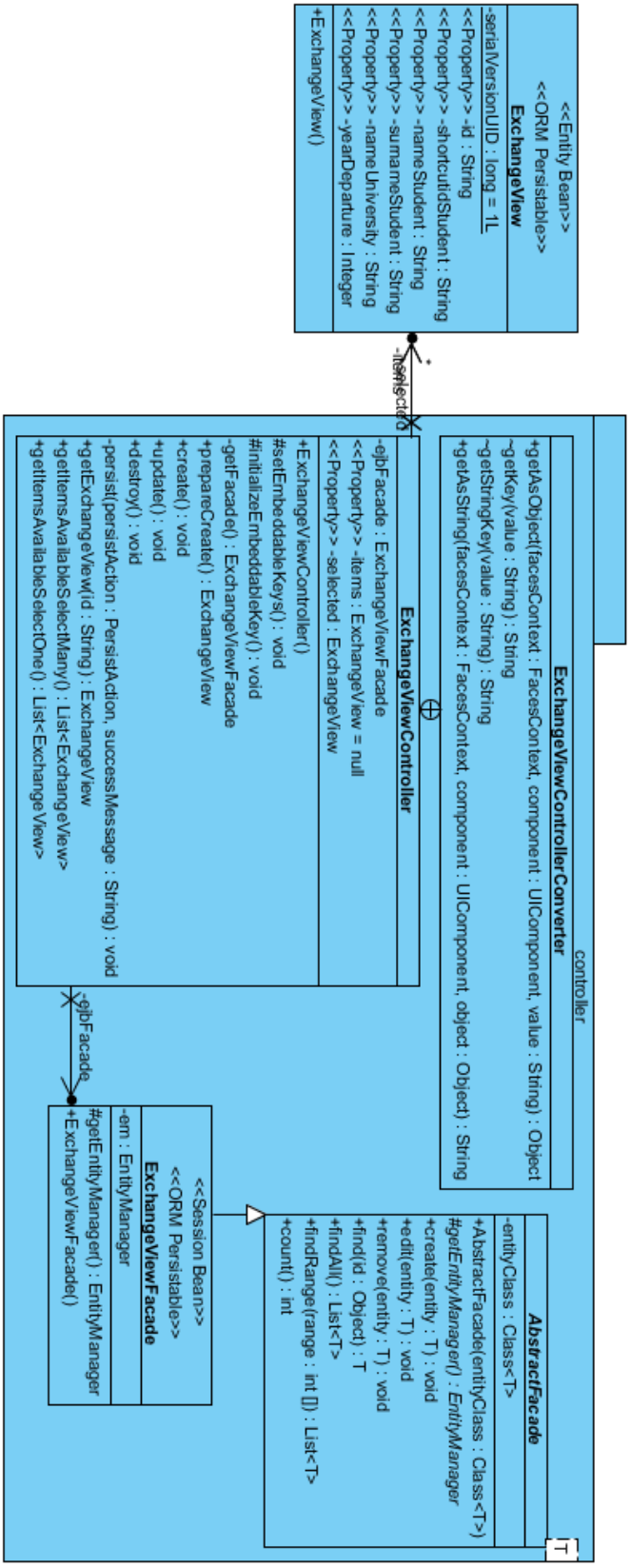
Příloha č. 1 – Diagramy tříd



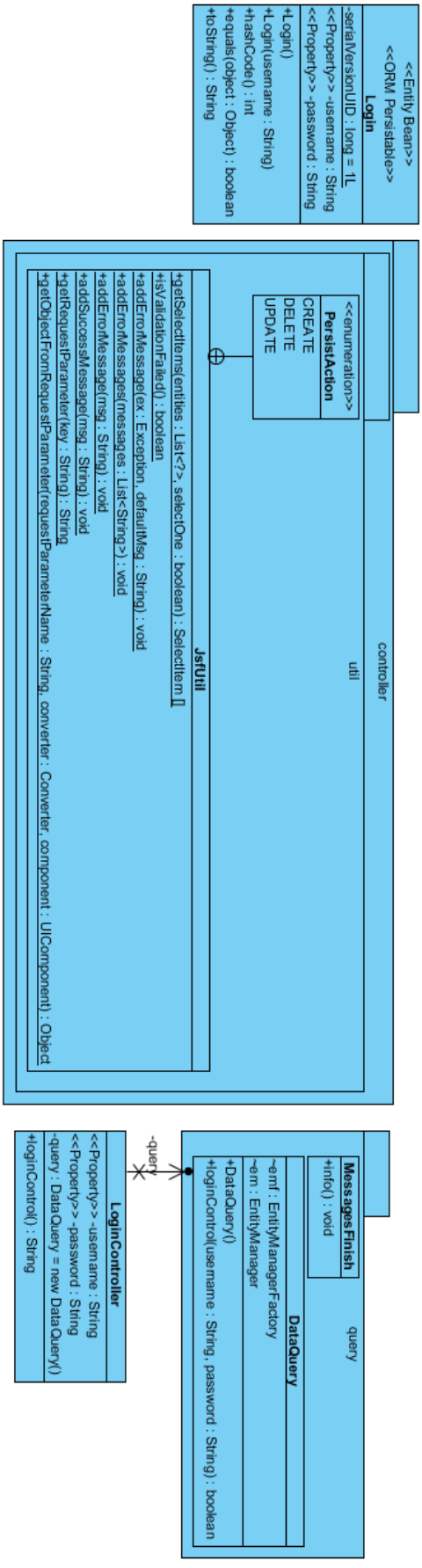
Obrázek 1 Class diagram entity Exam, jejich přidružených tříd a jejich vztahy



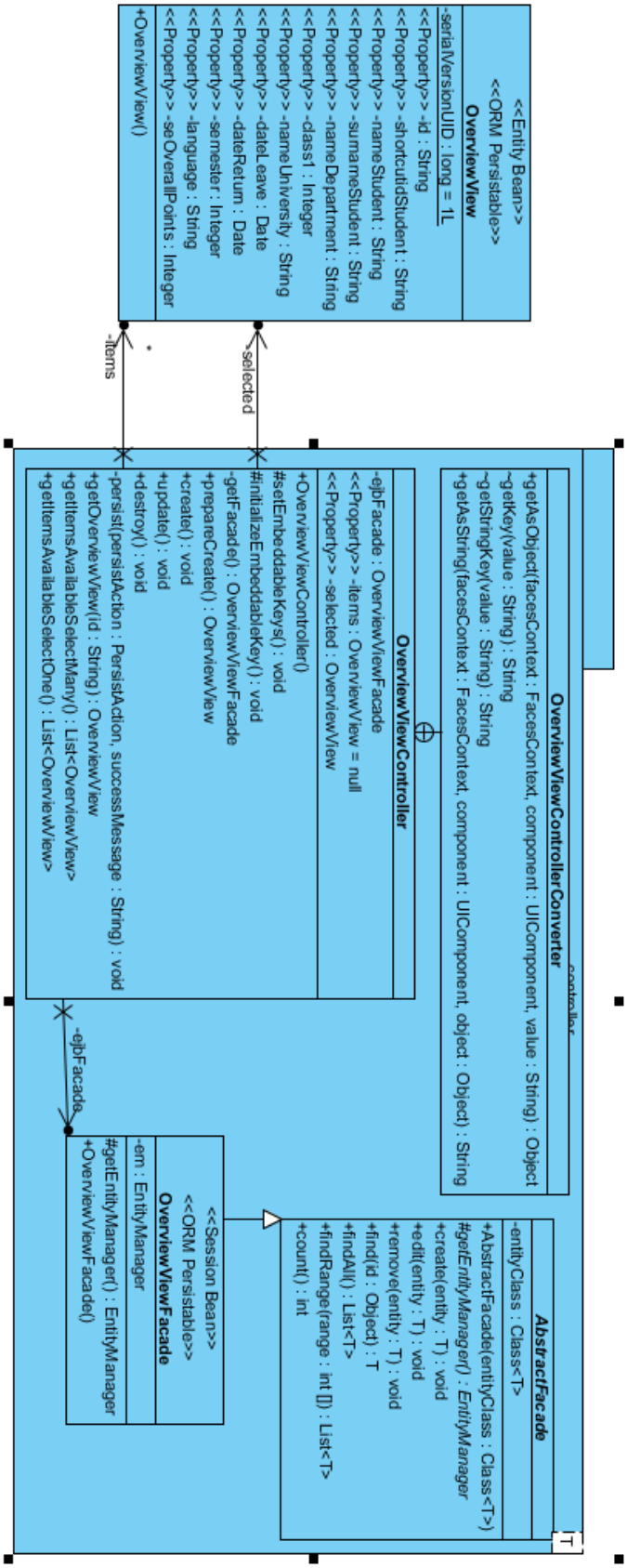
Obrázek 1 Class diagram entity Exchange, jejich přidružených tříd a jejich vztahy



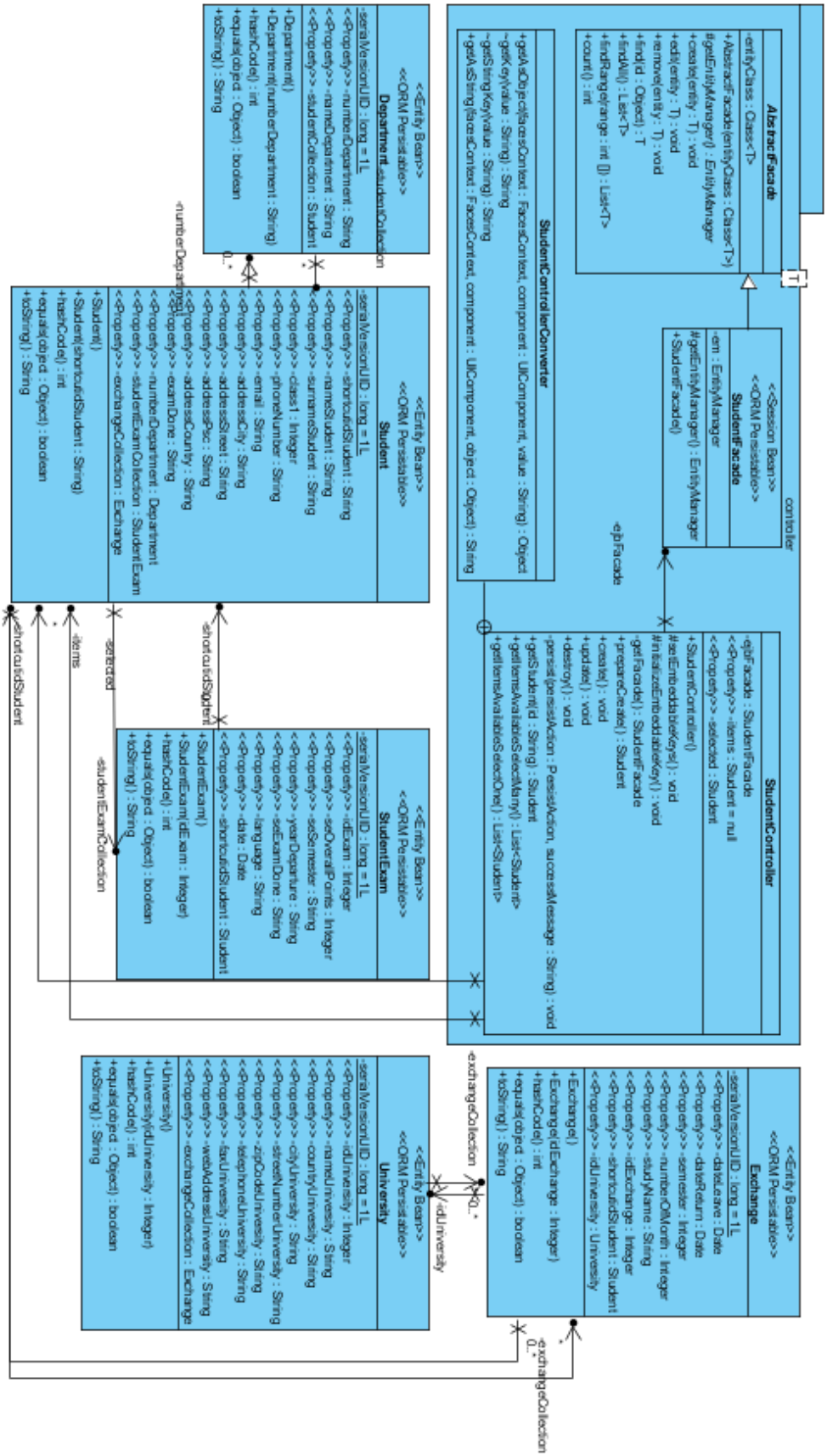
Obrázek 2 Class diagram entity ExchangeView, jejich přidružených tříd a jejich vztahy



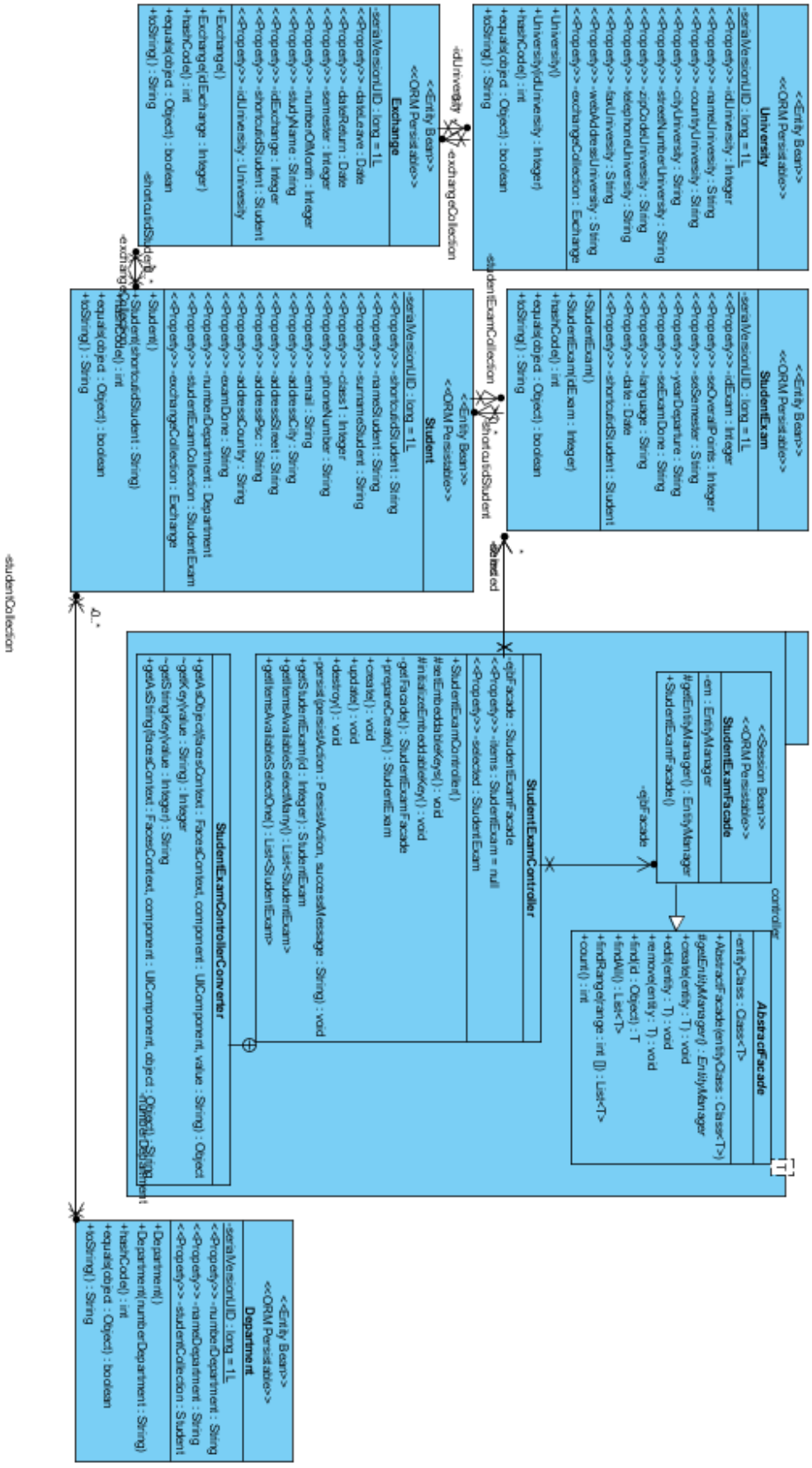
Obrázek 3 Class diagram entity Login, jejich přidružených tříd a jejich vztahy



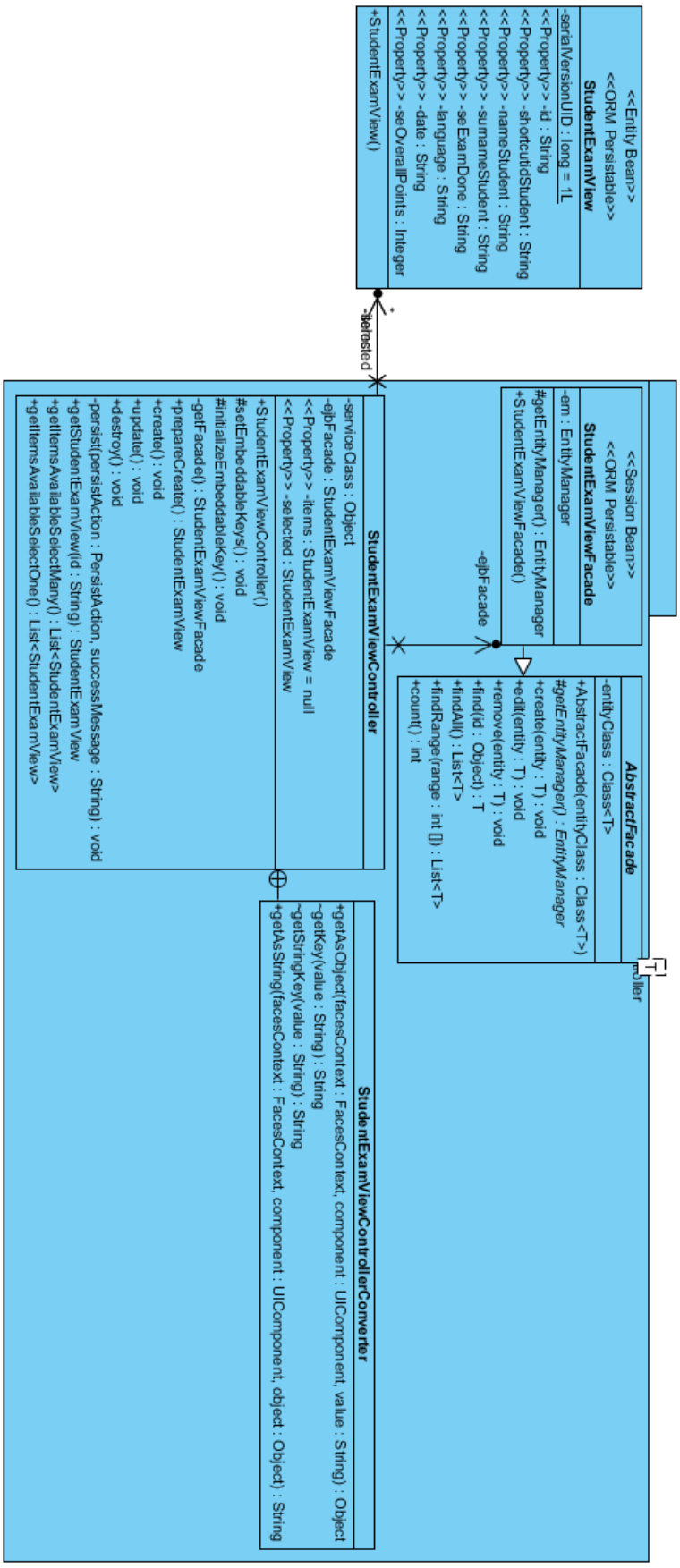
Obrázek 4 Class diagram entity OverviewView, jejich přidružených tříd a jejich vztahy



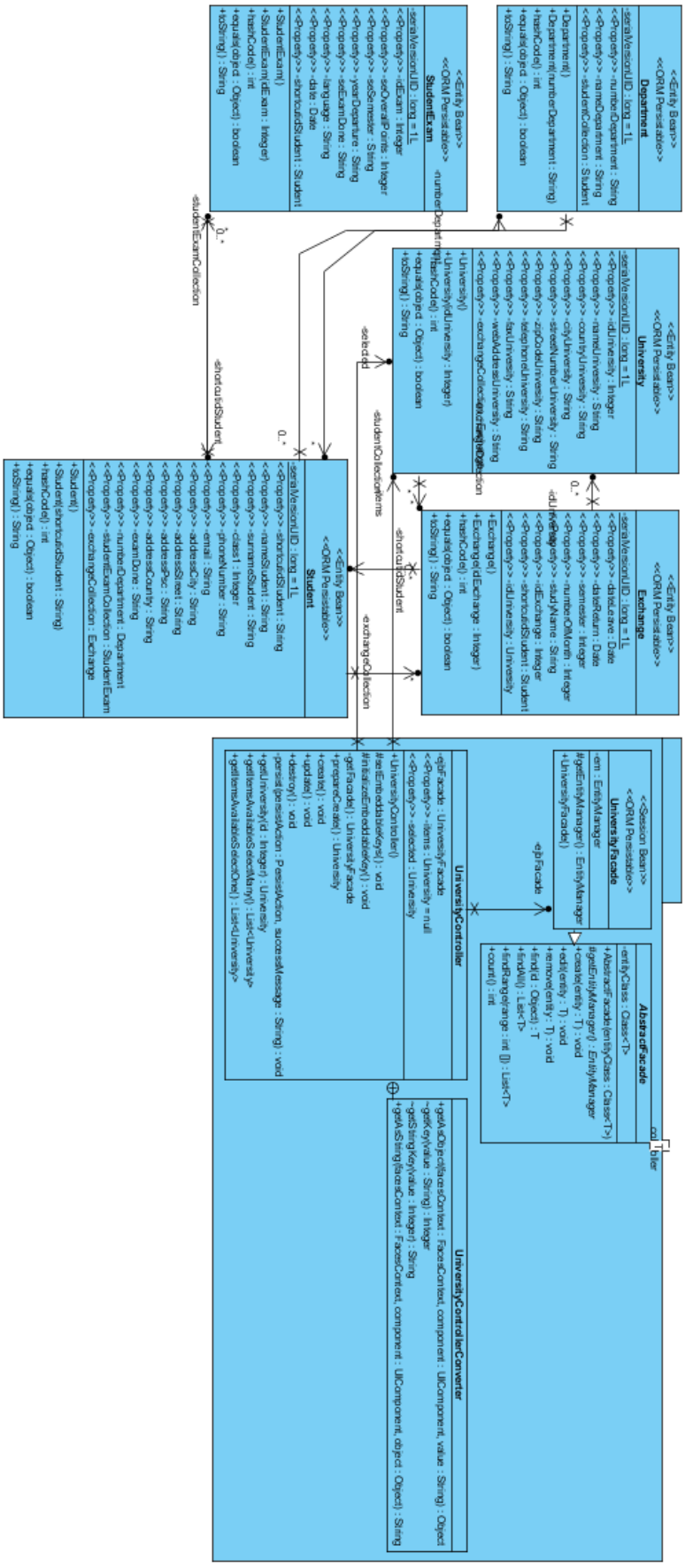
Obrázek 5 Class diagram entity Student, jejich přidružených tříd a jejich vztahy



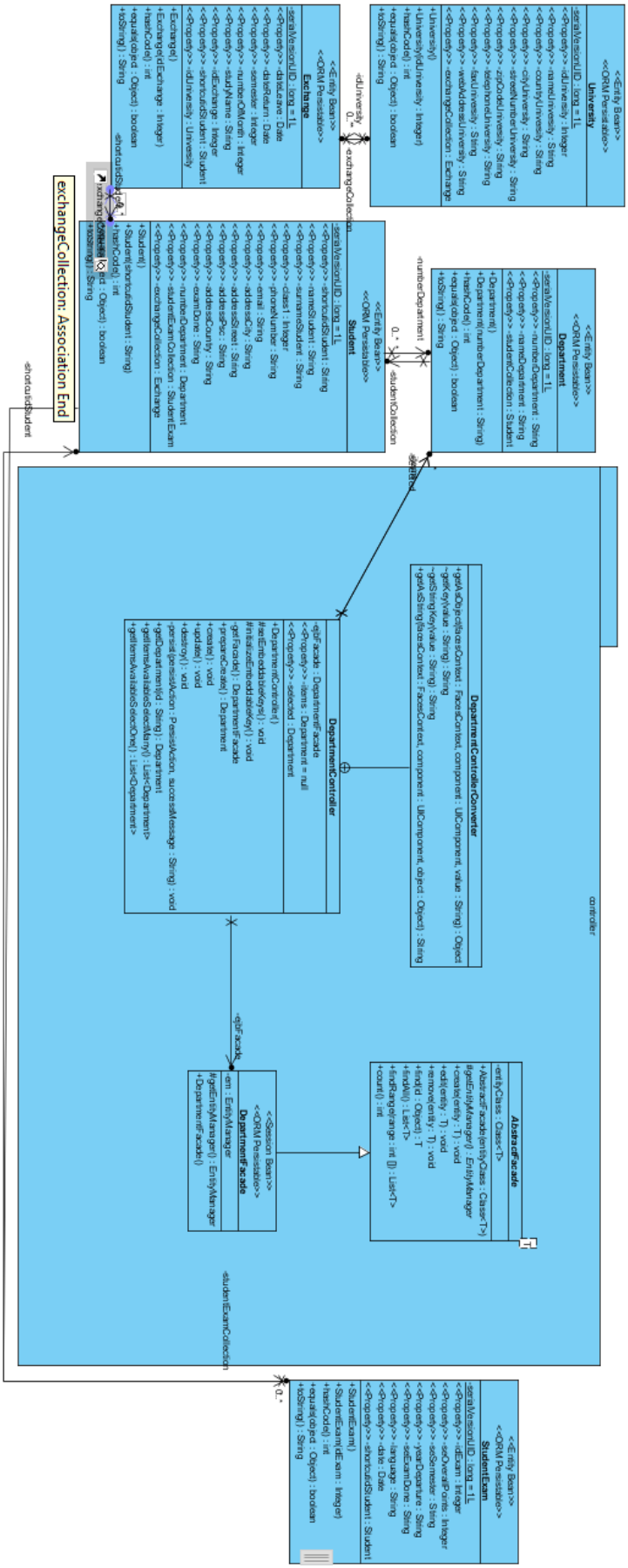
Obrázek 6 Class diagram entity StudentExam, jejich přidružených tříd a jejich vztahy



Obrázek 7 Class diagram entity StudentExamView, jejich přidružených tříd a jejich vztahy

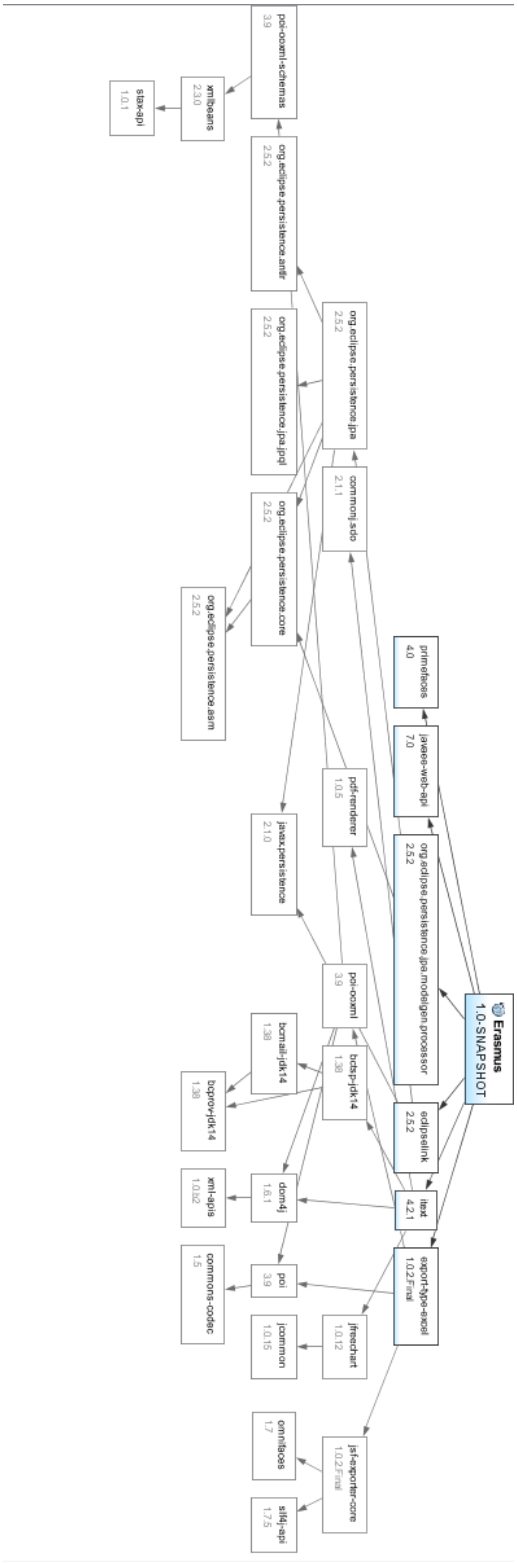


Obrázek 8 Class diagram entity University, jejich přidružených tříd a jejich vztahy



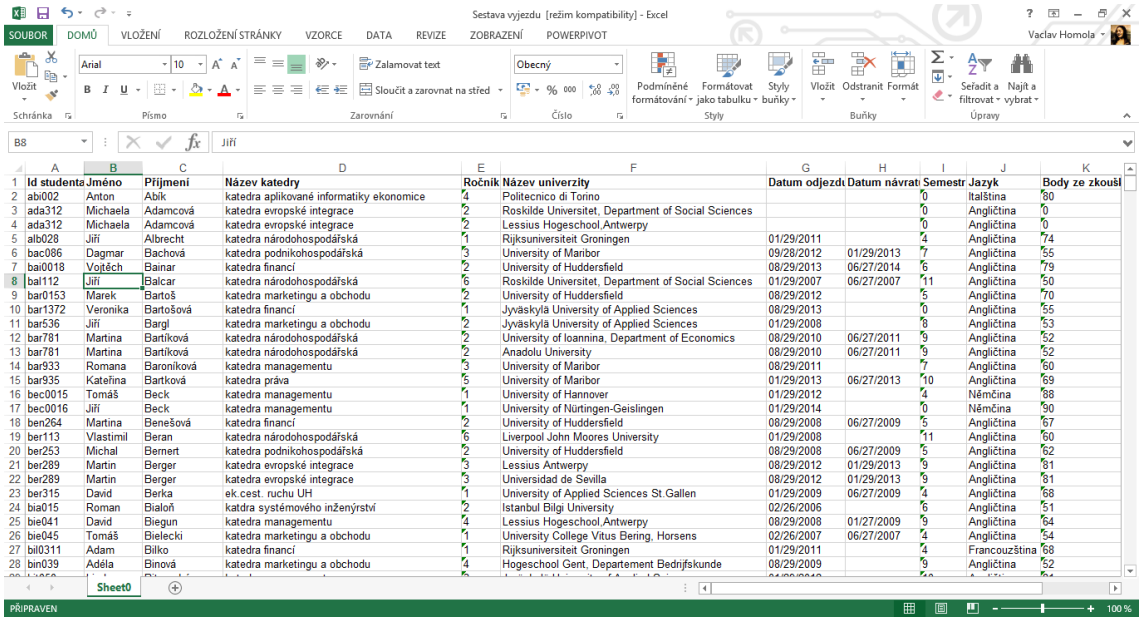
Obrázek 9 Class diagram entity Department, jejich přidružených tříd a jejich vztahy

Příloha č. 2 – Stromová struktura souboru pom.xml



Obrázek 11 Stromová struktura pom.xml

Příloha č. 3 – Výstup sestavy v excelu



	A	B	C	D	E	F	G	H	I	J	K
1	Id studenta	Jméno	Příjmení	Název katedry	Ročník	Název univerzity	Datum odjezdu	Datum návratu	Semestr	Jazyk	Body ze zkoušk
2	abi002	Anton	Abík	katedra aplikované informatiky ekonomice	4	Politecnico di Torino			0	Italština	80
3	ada312	Michaela	Adamcová	katedra evropské integrace	2	Roskilde Universitet, Department of Social Sciences			0	Angličtina	0
4	ada312	Michaela	Adamcová	katedra evropské integrace	2	Lessius Hogeschool,Antwerpy			0	Angličtina	0
5	alb028	Jiří	Albrecht	katedra národohospodářská	1	Rijksuniversiteit Groningen	01/29/2011		4	Angličtina	74
6	bac086	Dagmar	Bachová	katedra podnikohospodářská	3	University of Maribor	09/28/2012	01/29/2013	7	Angličtina	55
7	bai0018	Vojtěch	Bainar	katedra financí	2	University of Huddersfield	08/29/2013	06/27/2014	6	Angličtina	79
8	bal112	Jiří	Balcar	katedra národohospodářská	6	Roskilde Universitet, Department of Social Sciences	01/29/2007	06/27/2007	11	Angličtina	50
9	bar0153	Marek	Bartoš	katedra marketingu a obchodu	2	University of Huddersfield	08/29/2012		5	Angličtina	70
10	bar1372	Veronika	Bartošová	katedra financí	1	Jyväskylä University of Applied Sciences	08/29/2013		0	Angličtina	55
11	bar536	Jiří	Bargl	katedra marketingu a obchodu	2	Jyväskylä University of Applied Sciences	01/29/2008		8	Angličtina	53
12	bar781	Martina	Bartíková	katedra národohospodářská	2	University of Ioannina, Department of Economics	08/29/2010	06/27/2011	9	Angličtina	52
13	bar781	Martina	Bartíková	katedra národohospodářská	2	Anadolu University	08/29/2010	06/27/2011	9	Angličtina	52
14	bar933	Romana	Baroníková	katedra managementu	3	University of Maribor	08/29/2011		7	Angličtina	60
15	bar935	Kateřina	Bartková	katedra práva	5	University of Maribor	01/29/2013	06/27/2013	10	Angličtina	69
16	bec0015	Tomáš	Beck	katedra managementu	1	University of Hannover	01/29/2012		4	Němčina	88
17	bec0016	Jiří	Beck	katedra managementu	1	University of Nürtingen-Geislingen	01/29/2014		0	Němčina	90
18	ben264	Martina	Benešová	katedra financí	2	University of Huddersfield	08/29/2008	06/27/2009	5	Angličtina	67
19	ber113	Vlastimil	Beran	katedra národohospodářská	6	Liverpool John Moores University	01/29/2008		11	Angličtina	60
20	ber253	Michal	Bernert	katedra podnikohospodářská	2	University of Huddersfield	08/29/2008	06/27/2009	5	Angličtina	62
21	ber289	Martin	Berger	katedra evropské integrace	3	Lessius Antwerpy	08/29/2012	01/29/2013	9	Angličtina	81
22	ber289	Martin	Berger	katedra evropské integrace	3	Universidad de Sevilla	08/29/2012	01/29/2013	9	Angličtina	81
23	ber315	David	Berka	ek.cest. ruchu UH	1	University of Applied Sciences St.Gallen	01/29/2009	06/27/2009	4	Angličtina	68
24	bia015	Roman	Bialoň	katdra systémového inženýrství	2	Istanbul Bilgi University	02/26/2006		6	Angličtina	51
25	bie041	David	Biegun	katedra managementu	4	Lessius Hogeschool,Antwerpy	08/29/2008	01/27/2009	9	Angličtina	64
26	bie045	Tomáš	Bielecki	katedra marketingu a obchodu	1	University College Vitus Bering, Horsens	02/26/2007	06/27/2007	4	Angličtina	54
27	bil0311	Adam	Bilko	katedra financí	1	Rijksuniversiteit Groningen	01/29/2011		4	Francouzština	68
28	bin039	Adéla	Binová	katedra marketingu a obchodu	4	Hogeschool Gent, Departement Bedrijfskunde	08/29/2009		9	Angličtina	52

Obrázek 12 Výstup tabulek z aplikace do excelu